Centre for Quantum Technologies

NANYANG TECHNOLOGICAL UNIVERSITY

Research Topic | Improving existing magnetic sensors to measure state transition of ultra-cold atoms in micro-gauss regime

AMITY UNIVERSITY

<u>Performed at:</u>
Nanyang Technological University

<u>Project guide (name & sign):</u>

<u>Official seal:</u>

<u>Lead researcher (name & sign)</u>

TRIDIB BANERJEE

| MAE |

06/06/17 to 04/08/17

TABLE OF CONTENT

BACKGROUND |

My research mainly dealt with sensors that will be used to measure and control the magnetic fields to study coherent transmission of light through a cold atomic cloud. There are two types of sensors and both works on the same principle and is controlled via the same microcontroller. The only difference is few changes in the architecture of the PCB that they are integrated into. Our main element of the study is extremely cold strontium atoms (cooled to about 6 μK). It was experimentally determined that a magnetic field of 3.57 mG shifts the m = ±1 substates of the $^{88}Sr^{3}P_{1}$ excited state Γ = 2π = 7.5 kHz, which is its linewidth. This happens due to the Zeeman effect. This means that the atoms are particularly sensitive to the stray magnetic field that can be found in our laboratory. The stray field that should be compensated, includes the earth's magnetic field at typically 0.5 G. This is many orders of magnitude larger than the milligauss sensitivity of the intercombination transition. Besides, there is also the 50 Hz AC magnetic field from the line power supply. This has a peak-to-peak value of 1-2 mG in the lab. Human activities further introduce magnetic noise in the milligauss range. Unwanted magnetic field bias can also come from the components used in our experimental apparatus. To reduce this contribution, we mainly use non-magnetic components close to the science chamber. This will also lead to a more homogeneous field around the science chamber, thus improving the quality of the interpolation of the magnetic field at the position of the atoms by the sensor network. The remaining stray fields that remain are then compensated by an active control system. This system cancels the magnetic field below the level of 1 mG.

A flowchart summarizing the central system is shown in *Figure 1.1* It contains the following three major components,

- a sensor network to measure the magnetic field,
- a personal computer (PC), with a data acquisition card, for signal processing,
- several pairs of compensation coils to control the magnetic field.

The magnetic field is measured by a network of sensors. The sensor network consists of eight three-axis magnetic field sensors that are arranged in a cuboid geometry around the science chamber. The readings of the eight sensors are subsequently analyzed in a PC. The communication between the PC and the sensor network is mediated by a USB-CAN adapter. This adapter communicates with the sensor network through the controller area network (CAN) communication protocol, while it communicates with the PC through the universal serial bus (USB) communication protocol. The timing of the magnetic field measurement by each three-axis sensor is also synchronized by this adapter. In the PC, a user interface is implemented to operate the active system to control the stray magnetic field. It performs the digital processing of the measurement signals. When the magnetic field readings from the sensor network are received by the PC, the program performs an interpolation to find the magnetic field at the position of the trapped atoms. Several proportional-integral-derivative (PID) controllers and frequency filters are also implemented digitally to control the magnetic field. The control variable of the PID is converted by a data acquisition (DAQ) card (National Instruments PCI6733) to an analog voltage output, which is used to change the low noise current flowing through the compensation coils. This current is supplied by a current controller unit (Thorlabs LD8040). Two types of compensation coils are mounted. For the three components of the magnetic field, three pairs of Helmholtz coils are used to independently control the three components of the magnetic field. Three other pairs of coils are connected in the anti-Helmholtz configuration to control the $\frac{\partial Bx}{\partial x}$, $\frac{\partial By}{\partial y}$,and $\frac{\partial Bz}{\partial z}$, components of the field gradient.

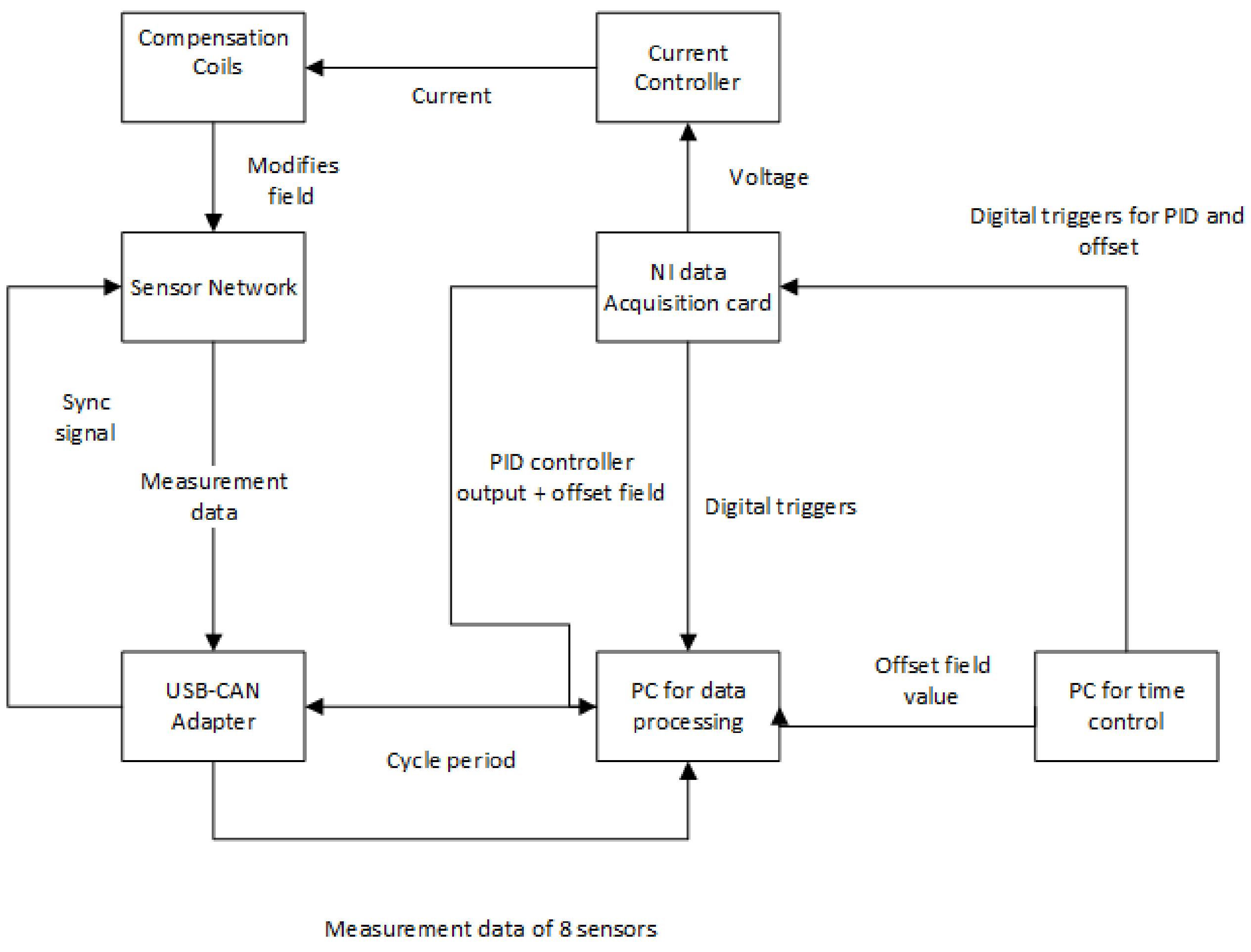


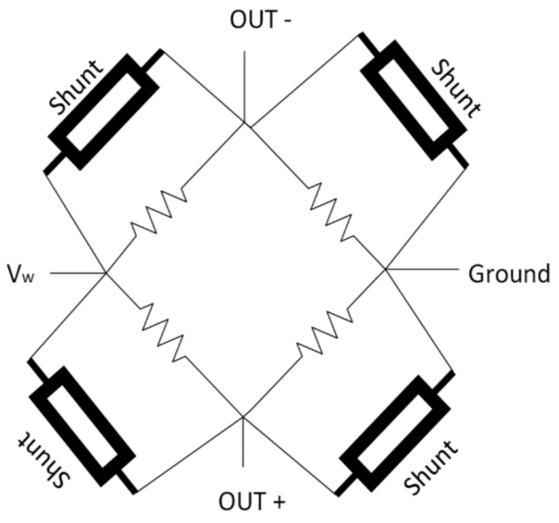
Figure 1.1: Flowchart summarizing the central system

SENSOR NETWORK |

The sensor network consists of eight three-axis sensors. Each three-axis magnetic field sensor consists of two commercial Honeywell magnetic field sensors. They are the HMC1001 and HMC1002 sensors, which are the single- and two-axis magnetic field sensors, respectively. The HMC sensors have analog measurement outputs, which re are subsequently amplified and converted to digital signals using an analog to digital converter (ADC, ADC7682).

HONEYWELL SENSOR |

The commercial Honeywell sensors, HMC1001, and HMC1002 are the components that are responsible for measuring the magnetic field. The HMC1001 sensor contains only one Wheatstone bridge, while in HMC1002, there are two Wheatstone bridges. The sensor operates based on the anisotropic magnetoresistive effect, with each Wheatstone bridge measuring one component of the magnetic field. By proper orientation of the HMC sensors on the printed circuit board (PCB) of the three-axis sensor, the full magnetic field vector can be measured. There are four resistive strips in each Wheatstone bridge; one strip on each arm of the bridge (see *Figure 1.2*). The resistive strips are made of nickel-iron (Permalloy) thin films. The sensor operates based on the AMR effect. Each permalloy thin film resistive strip is fabricated in a "barber pole" pattern, as shown in *Figure 1.3.* The "barber pole" configuration causes the current to flow at an angle of 45° along the resistive strip. The magnetization vector of the resistive strip is aligned with the easy axis of the resistive strip. Thus, the direction of the current through each of the resistive strip makes an angle of 45 with the magnetization vector. Ideally, when there is no external magnetic field, the bridge is



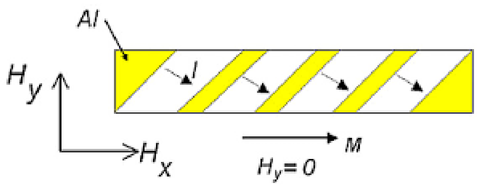


Figure 1.2: Configuration of Bridgestone bridge *Figure 1.3: Barber pole resistive strip*

balanced, i.e., all the resistive strips have equal resistance4. In reality, a zero balancing is never exactly achieved, leading to an offset in the output voltage of the bridge. The Wheatstone bridge in this configuration measures the magnetic field component that is along its sensitive axis which is perpendicular to the easy axis. When an external magnetic field B is applied along the sensitive axis of the Wheatstone bridge, a transverse magnetization component is added to each resistive strip. The angle between the current and the magnetization vector changes in a different manner for the different resistive strips. Since the magnetoresistance of the strip depends on the angle between the current and the magnetization vector, the balance of the Wheatstone bridge is disturbed. A voltage can now be measured across the two output pins of the Wheatstone bridge. For a small magnetic field, the change in the resistance is proportional to the component of the applied magnetic field B along the sensitive axis. The output voltage across the pins OUT+ and OUT- is

$$V_R = \frac{\Delta R}{R} V_w \propto B$$

where V_w is the voltage applied across the Wheatstone bridge. Each permalloy strip in an HMC sensor has many magnetic domains (see *Figure 1.4*).

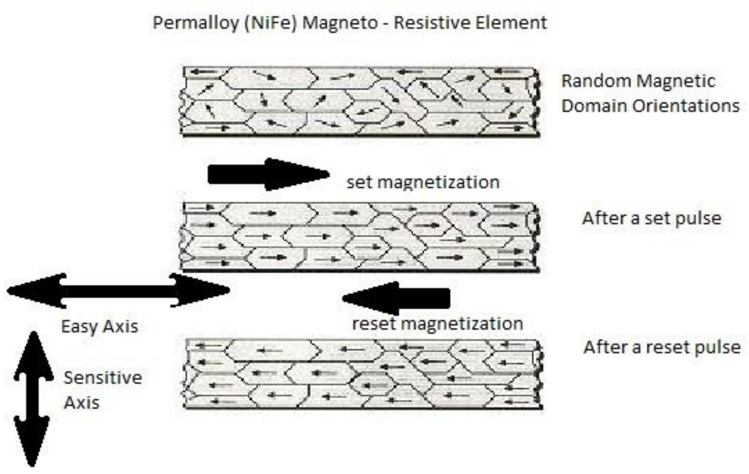


Figure 1.4: Magnetic domains for set and reset state

SET RESET STRAP |

When a large magnetic field is applied and removed subsequently, the magnetization vector may not return completely to its initial orientation along (or opposite) the easy axis. This results in a loss of sensitivity for the sensors. For the HMC sensor, a 3 G magnetic field is already sufficient to affect its sensitivity. In a typical cold atomic experiment, a large magnetic field is generated by the coils to carry out a magneto-optical trap (MOT). Since the sensors are in close proximity to the coils, the sensitivities of the sensors are affected by the preparation of the cold atomic cloud. To restore the sensor sensitivity, we use the set-reset straps of the HMC sensor. These set-reset straps apply a magnetic field to realign the magnetization vectors of the resistive strips. There are two possible directions to align the magnetization vector (see *Figure 1.4*), depending on the direction of the current in the set/reset

strap. If a positive current is flowing in the set-reset strap, the resistive strip is restored along the easy axis. This gives us the set state of the sensor. On the other hand, with a negative current, the resistive strip is put in the reset state. In the reset state, the magnetization vector is opposite to the easy axis. In both states, the sensor can be used to measure the magnetic field. Ideally, the sensitivities of the set and reset states are opposite in sign. A sketch of the expected sensor readout, as a function of the magnetic field component along the sensitive axis, is shown in *Figure 1.4*for the two states of the sensor. To study qualitatively the properties of the sensors, we place them inside a zero gauss chamber (ZGC). The ZGC consists of three layers of mu-metal magnetic shield. Inside the ZGC, we place a solenoid can be used to apply a uniform magnetic field. Ideally, the sensitivities of the set and reset states are opposite in sign. A sketch of the expected sensor readout, as a function of the magnetic field component along the sensitive axis, is shown in *Figure 1.5* for the two states of the sensor. To study qualitatively the properties of the sensors, we place them inside a zero gauss chamber (ZGC). The ZGC consists of three layers of mu-metal magnetic shield. Inside the ZGC, we place a solenoid can be used to apply a uniform magnetic field. The sensor is placed at the center of the ZGC and positioned such that the magnetic field generated by the solenoid is along its x-direction.

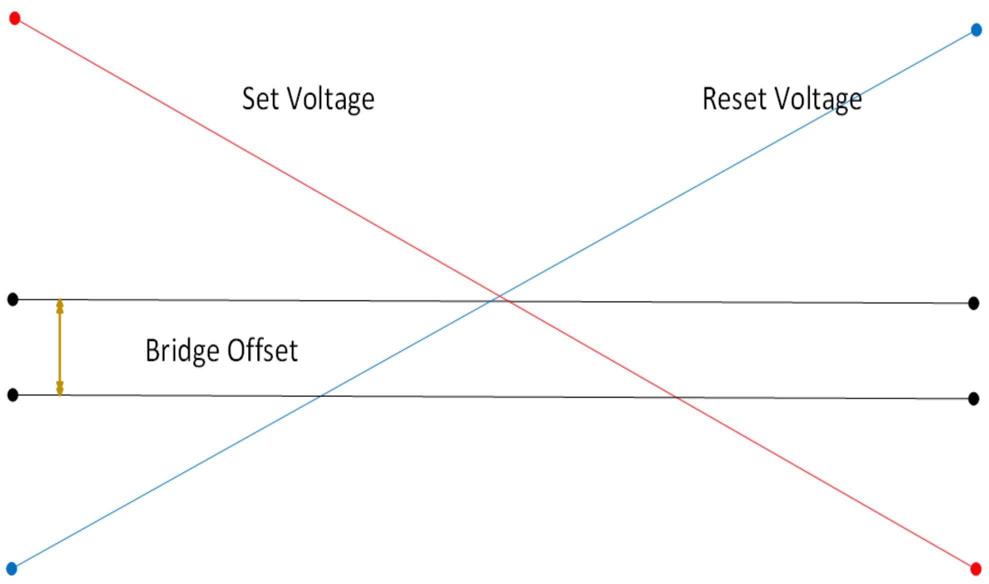


Figure 1.5: Relation between set, reset voltage and the bridge offset.

WORKING PRINCIPLE |

It works on the principle of Anisotropic Magnetoresistance. Magnetoresistance is the tendency of a material to change the value of its electrical resistance in an externally-applied magnetic field. There are a variety of effects that can be called magnetoresistance: some occur in bulk nonmagnetic metals and semiconductors, such as geometrical magnetoresistance, Shubnikov de Haas oscillations, or the common positive magnetoresistance in metals. Other effects occur in magnetic metals, such as negative magnetoresistance in ferro magnets or anisotropic magnetoresistance (AMR). Finally, in multicomponent or multilayer systems (e.g. magnetic tunnel junctions), giant magnetoresistance (GMR), tunnel magnetoresistance (TMR), colossal magnetoresistance (CMR), and extraordinary magnetoresistance (EMR) can be observed. Nowadays, systems are known (e.g. semimetals or concentric ring EMR structures) where a magnetic field can change resistance by orders of magnitude. As the resistance may depend on the magnetic field through various mechanisms, it is useful to separately consider situations where it depends on the magnetic field directly (e.g. geometric magnetoresistance, multiband magnetoresistance) and those where it does so indirectly through magnetization (e.g. AMR, TMR).

Thomson's experiments are an example of AMR, the property of a material in which a dependence of electrical resistance on the angle between the direction of electric current and direction of magnetization is observed. The effect arises from the simultaneous action of magnetization and spin-orbit interaction and its detailed mechanism depends on the material. It can be for example due to a larger probability of s-d scattering of electrons in the direction of magnetization (which is controlled by the applied magnetic field). The net effect (in most materials) is that the electrical resistance has a maximum value when the direction of current is parallel to the applied magnetic field. AMR of new materials is being investigated and magnitudes up to 50% have been observed in some ferromagnetic uranium compounds. In polycrystalline ferromagnetic materials, the AMR can only depend on the angle φ between the magnetization and current direction and it must follow,

$$\rho(\varphi)=\rho_{\text{perpendicular}} + (\rho_{\text{parallel}} + \rho_{\text{perpendicular}})\text{Cos}^2\varphi$$

where ρ is the (longitudinal) resistivity of the film and $\rho_{\text{perpendicular}}$, ρ_{parallel} are the resistivities for φ = 0° and 90° respectively. Associated with longitudinal resistivity, there is also transversal resistivity dubbed (somewhat confusingly) the planar Hall effect. To compensate for the non-linear characteristics and inability to detect the polarity of a magnetic field, the following structure is used for sensors. It consists of stripes of aluminum or gold placed on a thin film of permalloy (a ferromagnetic material exhibiting the AMR effect) inclined at an angle of 45°. This structure forces the current not to flow along the "easy axes" of the thin film, but at an angle of 45°. The dependence of resistance now has a permanent offset which is linear around the null point. Because of its appearance, this sensor type is called 'barber pole'.

The AMR effect is used in a wide array of sensors for measurement of Earth's magnetic field (electronic compass), for electric current measuring (by measuring the magnetic field created around the conductor), for traffic detection and for linear position and angle sensing. The biggest AMR sensor manufacturers are Honeywell, NXP Semiconductors, STMicroelectronics, and Sensitec GmbH.

THE SETUP PHASE :

OBJECTIVE 1 | THE INTERFACING PROGRAM |

The program is mainly written in C++ programming language. The program is responsible for several functions of which the primary ones are to accept data properly from the CAN adapter and to properly process it. The proper working of this program is essential to the entire project as every interaction with the sensor takes place through this program – the specifying and accepting of the buffer, the setting of the sample size, the setting of the sample period, the position, the file index – all is done through this program. The program just tells the sensors what to do after which they keep on sending data to the PC, constantly, till they are instructed to stop. It this interfacing PC program that decides how to interpret the incoming data, how many samples to consider, how fast to sample, how to store and transcode the incoming data into something portable and understandable and also most importantly, it is the program which helps us measure the magnetic field in all three axes. Not that the field values cannot be deduced from the raw data but without the proper transcoding of the byte stream by the program, no correct values could be deduced even from the raw data. We shall discuss the encoding part in a later section.

The program consists of 2 main parts:

- The CAN part
- The Computing part

The CAN part, named "candll" is responsible for proper interfacing with the CAN adapter. It contains libraries and headers which help us to properly interface with the required type of CAN adapter which is running a specific version of the firmware. It is also responsible for informing about the status of the CAN adapter – when it is disconnected, when it is not responding, etc. The computing part is responsible for accepting the byte stream from the CAN adapter, store it a buffer, properly decode the stream, sample it and henceforth, calculate and save the necessary parameters.

CONTRIBUTION |

- In the beginning, the program was erroneous and could not be debugged. It was then discovered that not only was it dependent on the type of the CAN adapter but also on the firmware version of the adapter. That is the candll libraries was not compatible with different CAN firmware.
- It was also discovered that the proper execution of the program also depended on the version of the visual studios (Visual Studios 15 enterprise edition) compiler used, the Windows development kit (Windows SDK) and the Microsoft Foundation Classes (MFC).

- Another thing that was discovered is that we explicitly needed to include some directories for the proper working of the program. A summary of directories explicitly included is given in ***Appendix A.1.***
- Another important thing that was observed was how the program was built around working with 8 sensors. It was not able to properly transcode or detect anything less than 8 sensors. This was rectified. All parts especially the ones where the magnetic field, the field gradients and the Maxwell identity was calculated for the three X, Y and Z axes, were modified such that they are able to deduce the right value (naturally with less precision) even with 1 sensor.
- Another important aspect was the reference to the Data Acquisition Card which was not used during the preliminary testing of the sensors because of which all reference to the DAC was to be removed. This turned out to be particularly hard but was done successfully.

OBJECTIVE 2 | THE ENCODING – DECODING SCHEME |

This was the biggest challenge in the interfacing program part of the research. Previously, the unipolar scheme was used by the sensors to send the data. The architecture of the new sensor was designed to facilitate the use of polar transfer scheme. Note that the old sensor was also capable of transferring in the polar scheme but the architecture and the interfacing program didn't permit it to operate in such a manner. This was to be changed and the new transfer scheme was to be incorporated in the new interfacing program.

It is required that information must be encoded into signals before it can be transported across communication media. In more precise words we may say that the waveform pattern of voltage or current used to represent the 1s and 0s of a digital signal on a transmission link is called digital to digital line encoding. There are different encoding schemes available:

1. Digital data to Digital signal
2. Digital data to Analog signal
3. Analog signal to Digital data
4. Analog signal to Analog data, etc.

The encoding of our present concern is Digital data to a digital signal. There are several ways to convert a digital data into digital signal:

1. Unipolar
2. Polar
3. Bipolar, etc

The scheme that was previously used is the unipolar scheme. Unipolar encoding uses only one level of value 1 as a positive value and 0 remains Idle. Since unipolar line encoding has one of its states at 0 Volts, it's also called Return to Zero (RTZ) as shown in *Figure 2.1.* A common example of unipolar line encoding is the 11'L logic levels used in computers and digital logic.

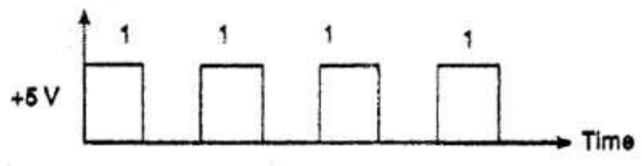


Figure 2.1: Unipolar encoding

Unipolar line encoding works well for inside machines - where the signal path is short - but is unsuitable for long distances, due to the presence of stray capacitance in the transmission medium. On long transmission paths, the constant level shift from 0 to 5 volts, which causes the stray capacitance to charge up. There will be a "stray" capacitor effect between any two conductors that are in close proximity to each other. For example, parallel running cables or wires are very susceptable to stray capacitance.Consider *Figure 2.2*

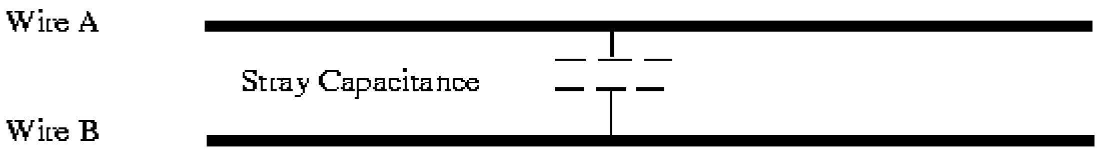


Figure 2.2: Stray capacitance between any two cables in close proximity

The dc component will cause the capacitor plates to charge up. See *figure 2.3 and 2.4.*

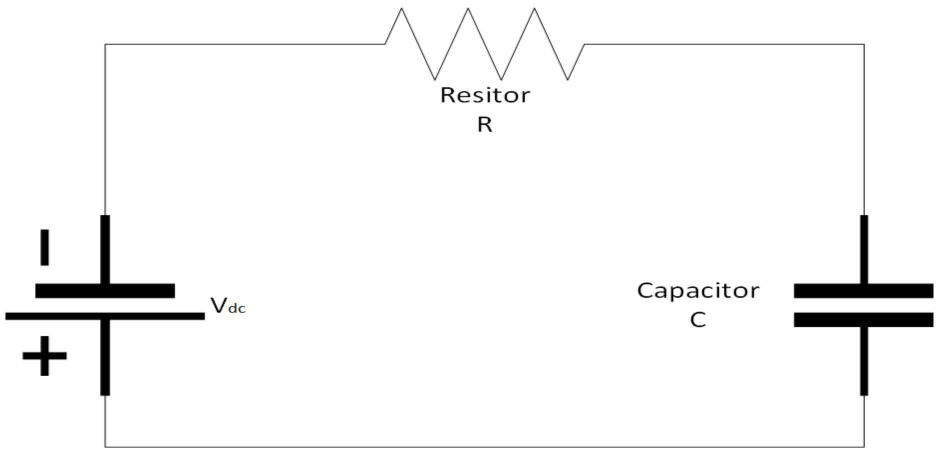


Figure 2.3: Charging of capacitor

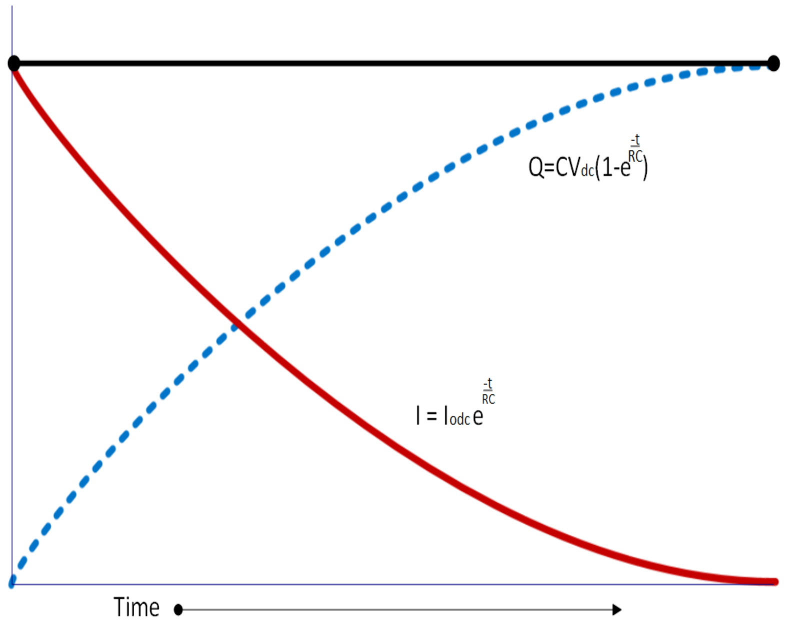


Figure 2.4: Charging current and voltage on capacitor plate

If there is sufficient capacitance on the line (and a sufficient stream of 1s) a DC voltage component will be added to the data stream. Instead of returning to 0 volts, it would only return to 2 or 3 volts. See *figure 2.5.* The receiving station may not recognize a digital low at a voltage of 2 volts.

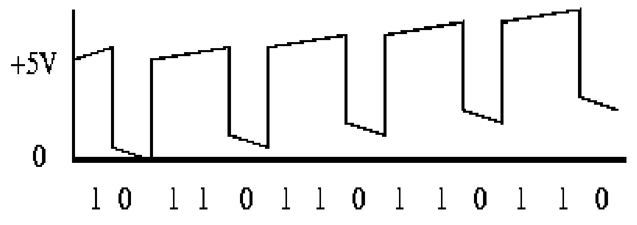


Figure 2.5: DC component problem in unipolar encoding scheme

Unipolar line encoding can also have synchronization problems between the transmitter and receiver's clock oscillator. The receiver's clock oscillator locks onto the transmitted signal's level shifts (logic changes from 0 to 1) if there is a long series of logical 1s or 0s in a row. There is no level shift for the receiver's oscillator to lock to. The receiver oscillator's frequency may drift and become unsynchronized: it could lose track of where the receiver is supposed to sample the transmitted data.

In the polar scheme, the digital encoding is symmetrical around 0 Volts. The signal does not return to zero; it is either a +ve voltage or a -ve voltage. Polar line encoding is also called None Return to Zero (NRZ). Polar line encoding is the simplest pattern that eliminates most of the residual DC problem. There is still a small residual DC problem, but Polar line encoding is a great improvement over Unipolar line encoding. Polar encoding has an added benefit in that it reduces the power required to transmit the signal by one-half. Polar and Unipolar line encoding both share the same synchronization problem though: if there is a long string of logical 1s or 0s, the receive oscillator may drift and become unsynchronized.

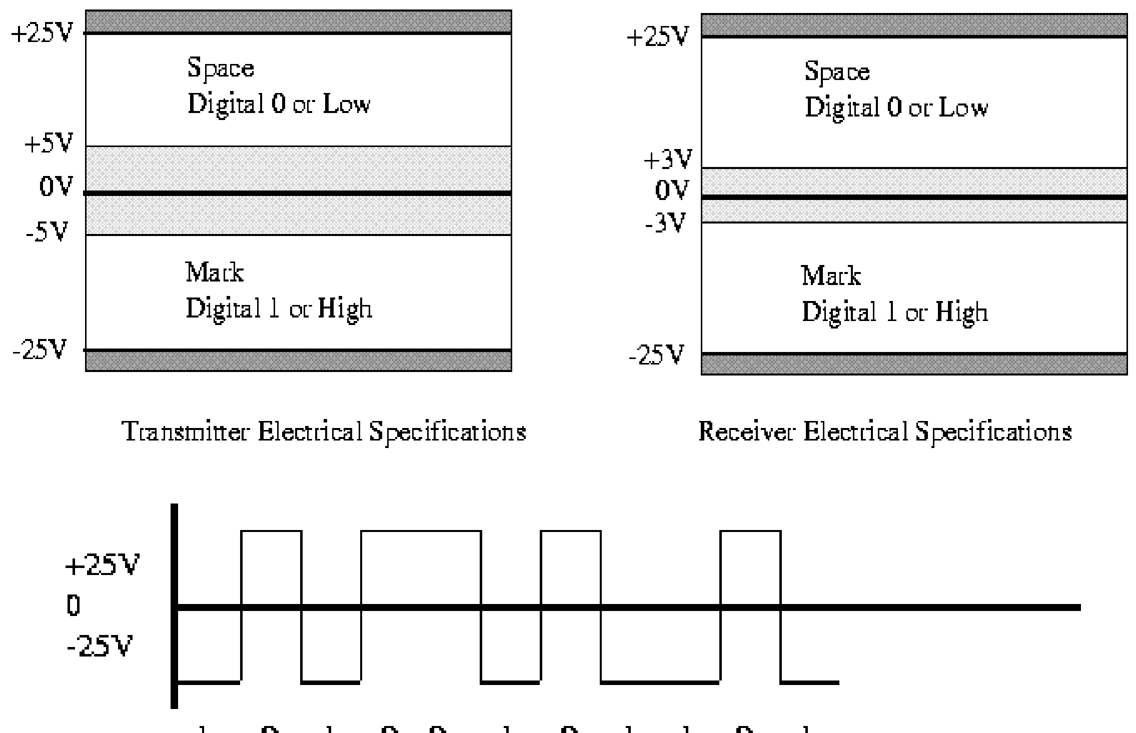


Figure 2. 6: Polar encoding scheme

CONTRIBUTION I

The first problem was to properly understand where and how will the change affect the working of the sensor, the communication channel, the CAN adapter and the PC interfacing program itself.

The very first fundamental change comes in the how the data is handled by the PC interface program. The buffer here must accept data in both positive and negative regimes. Typically, the data array for the buffer is defined as an unsigned short int data type. This means that each element of the data array could accept values of digital level from 0 to 65,535. However, this needed to be changed into signed short int data type. It could now accept values of digital level from 2,768 to 32,767. While the range remains same, now it should be distributed symmetrically on both sides of the origin. Given in ***Appendix B.1*** are the parts of the code in the PC interface program that needed to be changed.

Things also needed to be changed on a much more fundamental level. The sensor's firmware also needed to be updated to make it compatible with the polar encoding scheme. In this section, we only deal with the changes to the firmware that affects the encoding scheme. In a later section, the firmware will be explained in details. Given in ***Appendix B.2*** are the parts of the code from the sensor's firmware that were changed to make it compatible with the new encoding scheme.

OBJECTIVE 3 | THE SENSOR FIRMWARE |

The firmware of the sensor is one of the critical components without which the sensor won't work no matter what. The biggest problem in this seemingly easy piece of the program was that the firmware was originally written in an extremely nonportable manner. Not only was the program sensitive to the version of the platform used to write the code, it was also sensitive to the version of the platform and more importantly, the version and edition of the operating system that was used to write the program on. This was a huge problem – to figure out how to make it executable again.

CONTRIBUTION |

The very first challenge was to determine the right combination of platform, operating system, and other details for execution of the program. Given in ***Appendix C.1*** is the proper configuration.

This was just the first step. The next equally critical part was to find the proper CAN library. The firmware uses the headers and classes in the CAN library to enable the sensor to be able to properly communicate data to the CAN. The problem was the CAN was changed multiple over the course of the project. This caused a bottleneck in trying to remake the firmware. There arose three possibility – the sensor couldn't communicate at all due to the import of the wrong library, the sensor could communicate with the CAN but not with maximum accuracy – that it would understand basic commands but the data send by the sensor would often be misinterpreted and everything from there would only go on to get further corrupt. The third possibility was the hardest one to reach to – the perfect CAN library was to be found and a CAN adapter that runs firmware using this library was also to be found.

The next step was to determine the correct FUSE settings to make the sensors work. The problem was that these FUSE settings could result in everything from the sensor being undetectable to the sensor simply not responding or even worse – giving wrong values. Unless one is absolutely sure about the authenticity of the fuse settings, there was no way to differentiate between a sensor with wrong FUSE settings and a sensor which is actually defective. To add to the problem, the sensors were different to those of the previous one thus the FUSE settings from the previous ones just couldn't be copied. To make matters worse, the new sensors didn't have any FUSE settings by default at all. Over course of extensive hit and trails, the FUSE settings given in ***Appendix C.2*** were found to be the perfect ones.

THE EXPERIMENTATION PHASE :

Once the sensors could be successfully detected, their analysis began. Initially, they were attached one by one, to the CAN adapter and were used to take a reading of the magnetic field over a short period of time. The important specification of the configuration is given below:

- Sampling period: 2048 μs
- Number of samples: 33,000
- Write buffer index: 1

The sampling period is the time over which a single sample consisting of a set and reset value is taken and written into the buffer. To understand the how a sampling of the set and reset values are done, what shout be the correct time, etc, we must first understand the circuit of the sensor and how it works. Remember that the sensor needs the magnetic domains to be realigned to maintain its sensitivity. Given below are the pin diagram of the sensors:

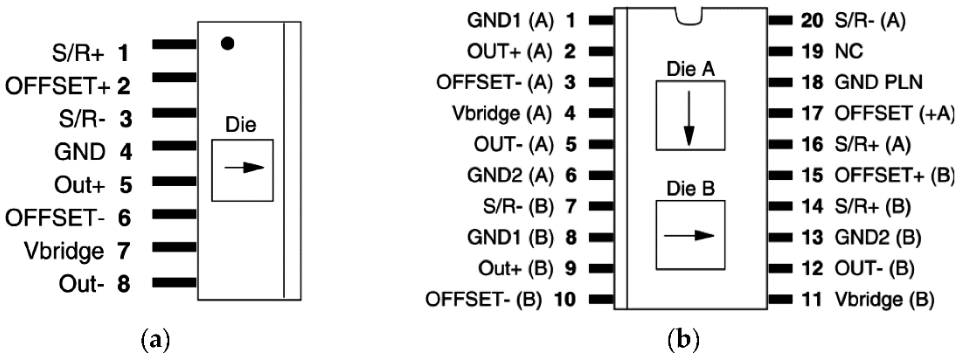


Figure 3.1 Pin configuration of old(a) and new(b) sensors

Currently, only the new ones are used. Their industrial name is HMC 1002 Honeywell sensors. As we can see they have dedicated set-reset pins. The Older sensor has 1 dedicated pin, pin no 1 and sensor the newer sensor has 4 pins for set & reset – pin no 7,14,16 and 20. These pins are used to send positive and negative pulses for the set and reset. Most low field magnetic sensors will be affected by large magnetic disturbing fields (>4 - 20 gauss) that may lead to output signal degradation. In order to reduce this effect, and maximize the signal output, a magnetic switching technique can be applied to the MR bridge that eliminates the effect of past magnetic history. The purpose of the Set/Reset (S/R) strap is to restore the MR sensor to its high sensitivity state for measuring magnetic fields. This is done by pulsing a large current through the S/R strap. The Set/Reset (S/R) strap looks like a resistance between the SR+ and SR- pins. This strap differs from the OFFSET strap in that it is magnetically coupled to the MR sensor in the cross-axis, or insensitive, direction. Once the sensor is set (or reset), low noise and high sensitivity field measurement can occur. In the discussion that follows, the term “set” refers to either a set or reset current.

When MR sensors exposed to a magnetic disturbing field, the sensor elements are broken up into randomly oriented magnetic domains that lead to sensitivity degrading. A current pulse (set) with a peak current above minimum current in spec through the Set/Reset strap will generate a strong magnetic field that realigns the magnetic domains in one direction. This will ensure a high sensitivity and repeatable reading. A negative pulse (Reset) will rotate the magnetic domain orientation in the opposite direction, and change the polarity of the sensor outputs. The state of these magnetic domains can retain for years as long as there is no magnetic disturbing field present. The on-chip S/R should be pulsed with a current to realign, or "flip", the magnetic domains in the sensor. This pulse can be as short as two microseconds and on average consumes less than 1 mA dc when pulsing continuously. The duty cycle can be selected for a 2 μsec pulse every 50 msec, or longer, to conserve power. The only requirement is that each pulse only drives in one direction. That is, if a +3.5 amp pulse is used to "set" the sensor, the pulse decay should not drop below zero current. Any undershoot of the current pulse will tend to "un-set" the sensor and the sensitivity will not be optimum.

Using the S/R strap, many effects can be eliminated or reduced that include: temperature drift, nonlinearity errors, cross-axis effects, and loss of signal output due to the presence of high magnetic fields. This can be accomplished by the following process:

- A current pulse, Iset, can be driven from the S/R+ to the S/R- pins to perform a "SET" condition. The bridge output can then be measured and stored as Vout(set).
- Another pulse of equal and opposite current should be driven through the S/R pins to perform a "RESET" condition. The bridge output can then be measured and stored as Vout(reset).
- The bridge output, Vout, can be expressed as:Vout = [Vout(set) - Vout(reset)]/2. This technique cancels out offset and temperature effects introduced by the electronics as well as the bridge temperature drift.

There are many ways to design the set/reset pulsing circuit, though, budgets and ultimate field resolution will determine which approach will be best for a given application.

Given below is the set-reset characteristics from the manufacturer's datasheet for the sensor

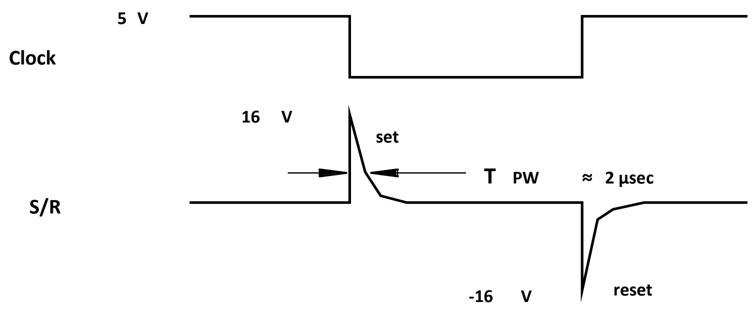


Figure 3.2: Set reset timing

CONTRIBUTION |

Using an oscilloscope, across pins 20 and 16 for the new one and pins 1 and 3 for the old one, the set-reset pulses of the sensors were experimentally measured. Given in ***Appendix D.1*** is the program that was executed in Matlab to analyze the data from the oscilloscope, followed by the results in *Figure 3.3, 3.4, 3.5, and 3.6.*

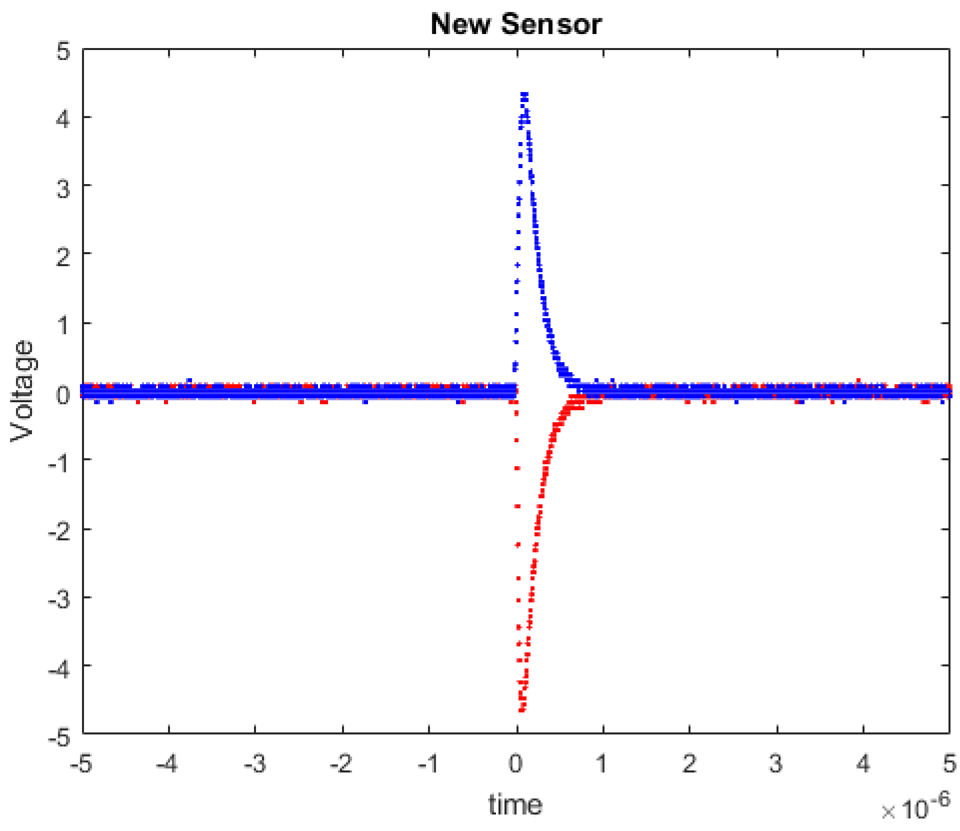


Figure 3.3: Set – Reset characteristics of the new sensors

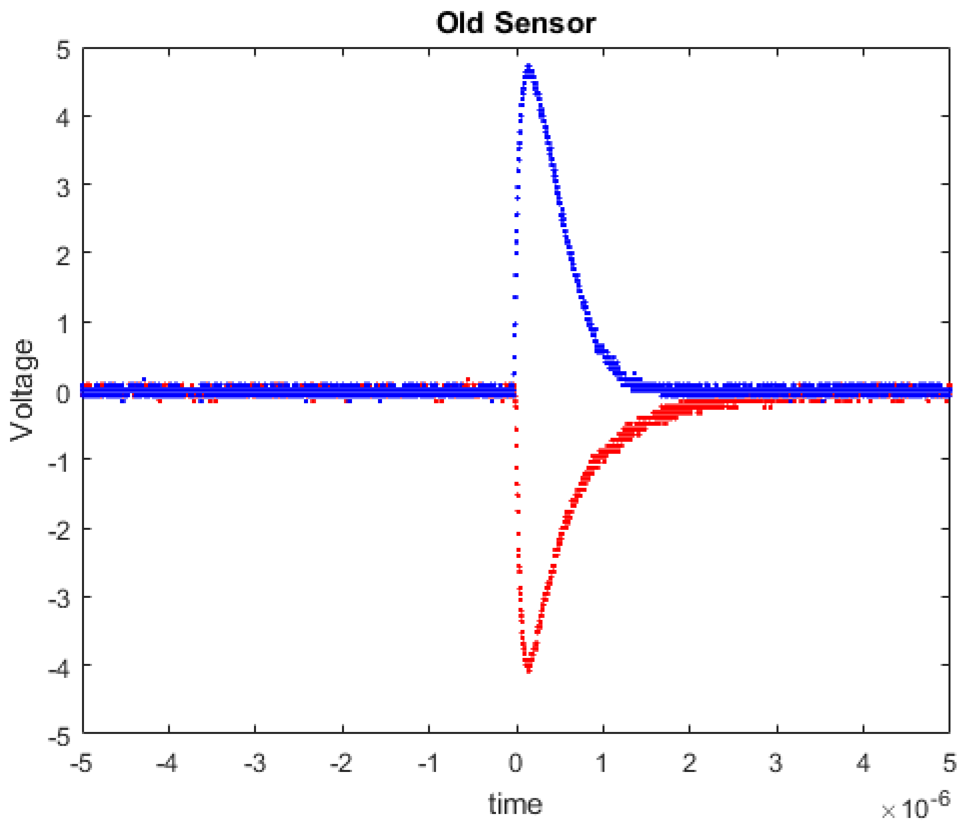


Figure 3.4: Set – Reset characteristics of the old sensors

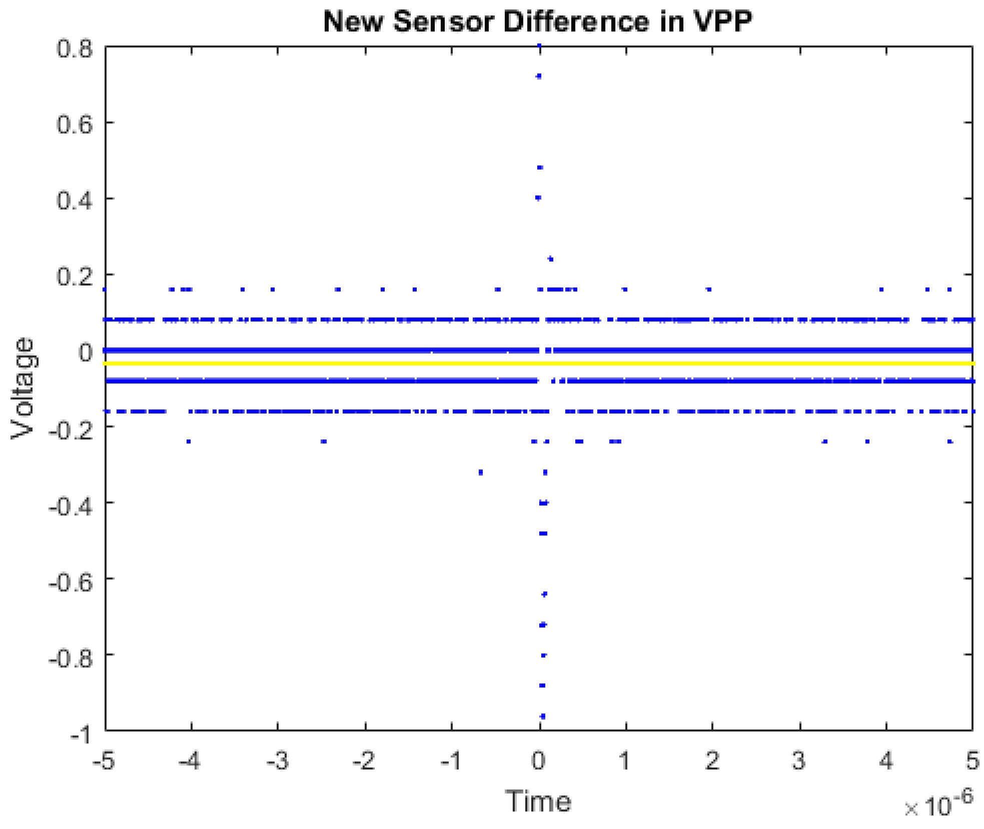


Figure 3.5: Set – Difference in Set – Reset sensitivity of new sensor

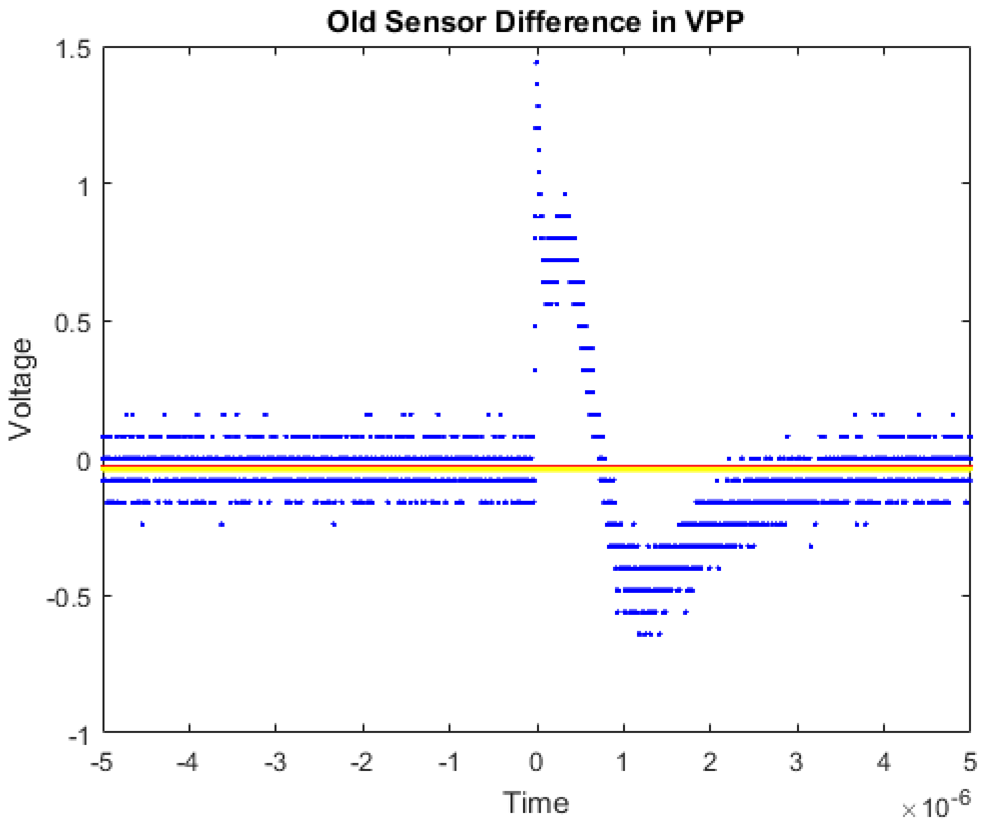


Figure 3.6: Set – Difference in Set – Reset sensitivity of old sensor

OBSERVATIONS |

Following were the observations:

1. The set-reset pulses of the new sensors were similar and took 0.796 µsec with a tolerance of ± 0.168 µsec.
2. The set pulse of the old sensors took 1.456 µsec with a tolerance of ± 0.204 µsec.
3. The reset pulse of the old sensors took 1.928 µsec with a tolerance of ± 0.224 µsec.
4. The set and reset pulses of the old sensors were not similar, the resetting taking much more time than the setting.
5. For the old sensor, $\frac{Vset}{Vreset}$ = -1.1569 (i.e. 15.69% change)
6. For the old sensor, $\frac{Vset}{Vreset}$ = -0.9310 (i.e. 6.90 % change)

However, the above shown graphical representations should not be mistaken for a common timeline representation. The timescale was adjusted to further enhance the difference in the set-reset symmetry. When displayed on a common timeline, they look something like *figure 3.6 and 3.8.*

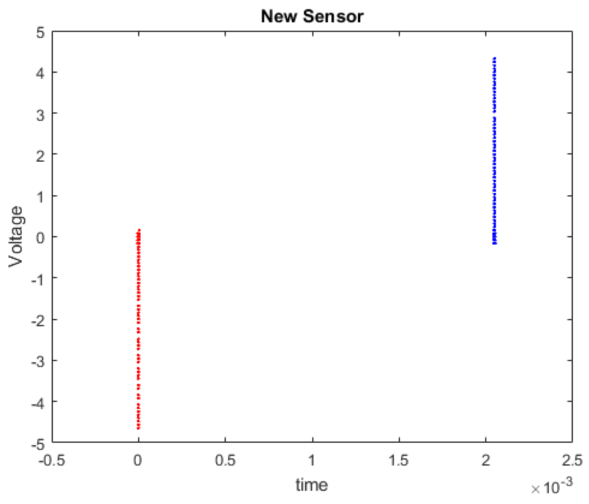


Figure 3.7: Set-Reset Pulse of new sensor represented on a common timeline

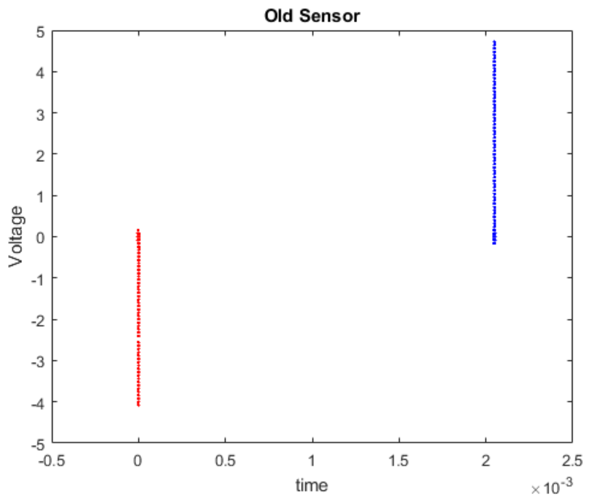


Figure 3.8: Set-Reset Pulse of old sensor represented on a common timeline

As we can see, the set reset pulses are separated by a time of 2048 µsec. The sensors give analog voltage output. That means after set pulse hits the sensor, the magnetic domains are aligned along a specific axis with respect to which it measures the value of the magnetic field in form of voltage. This voltage is available across the output pins till the reset pulse hits. The reset pulse realigns the domains in an opposite direction and now the sensor gives voltage output accordingly in the output pins for the next 2048 µsec till yet another set pulse hits. Thus, rather than trying to sample during the pulses itself (which will require a sampling frequency of 1Mhz) we can sample it during these 2048 µsec gaps which brings our required sampling frequency down to 488.28 Hz. The Nyquist frequency thus becomes 244.14 Hz. This is how data is sampled.

Before measuring the sensitivity, we also needed to measure the impedance of the zero gauss chamber and its nature. The impedance of the coil in the zero gauss chamber was calculated to be 3.968 ± .021 Ω. All the sensors, when put into the zero gauss chamber would be exposed to an applied sinusoidal magnetic field of frequency 100 mHz. the impedance of the coil was tested from 0 Hz to 1000 mHz, but the impedance remained almost the same suggesting that the impedance was mostly resistance. This means it would be safe to assume the impedance to be constant during our experimentation with the sensors when they are put inside the zero gauss chamber.

OBJECTIVE 4 | TO DETERMINE THE SENSITIVITY | CONTRIBUTION |

To determine the sensitivity, the sensors were first mounted on a fixture. This was done to ensure that the x axis of the sensor was coincident with the axis of the magnetic coil of the ZGC. After this, the sensors were placed inside the chamber one by one. They were subjected to a magnetic field of varying magnitude (but constant frequency) until the point they were just about to saturate. They were then made to take 33000 samples of the field at this point. Given below are the details of the applied magnetic field. The program written to analyze the data is in ***Appendix E.1*** and the results in ***Appendix E.2***.

1. Applied sinusoidal Voltage: 2 to 6 Volts
2. Time period: 10 seconds
3. Frequency: 100mHz
4. Conversion factor: 12.84 Gauss/Volts applied
5. Coil impedance: 3.968 ± .021 Ω.

Short_n=50000

Device ID	Short_Vpp(V)	Short_Vcc(mV)	Short_Current(mA)	Field (mG)	Label
S17	6	448	109.268	1.4078	data00000
S18	-	-			
S7	5	376	91.707	1.1775	data00001
S5	5	376	91.707	1.1775	data00002
S14	4	296	72.439	0.9301	data00003
S10	6	452	110.244	1.4155	data00004
S4	-	-			
S6	4	300	73.171	0.9395	data00005
S8	6	448	109.268	1.403	data00006
S1	5	384	93.659	1.2026	data00007
S11	-	-			
S16	5	376	91.707	1.1775	data00008
S2	4	296	72.195	0.9269	data00009
S12	2	156	38.049	0.4885	data00010
S3	2	156	38.049	0.4885	data00011
S15	6	452	110.244	1.4155	data00012
S13	6	452	110.244	1.4155	data00013

The X and Y axis both are digital levels

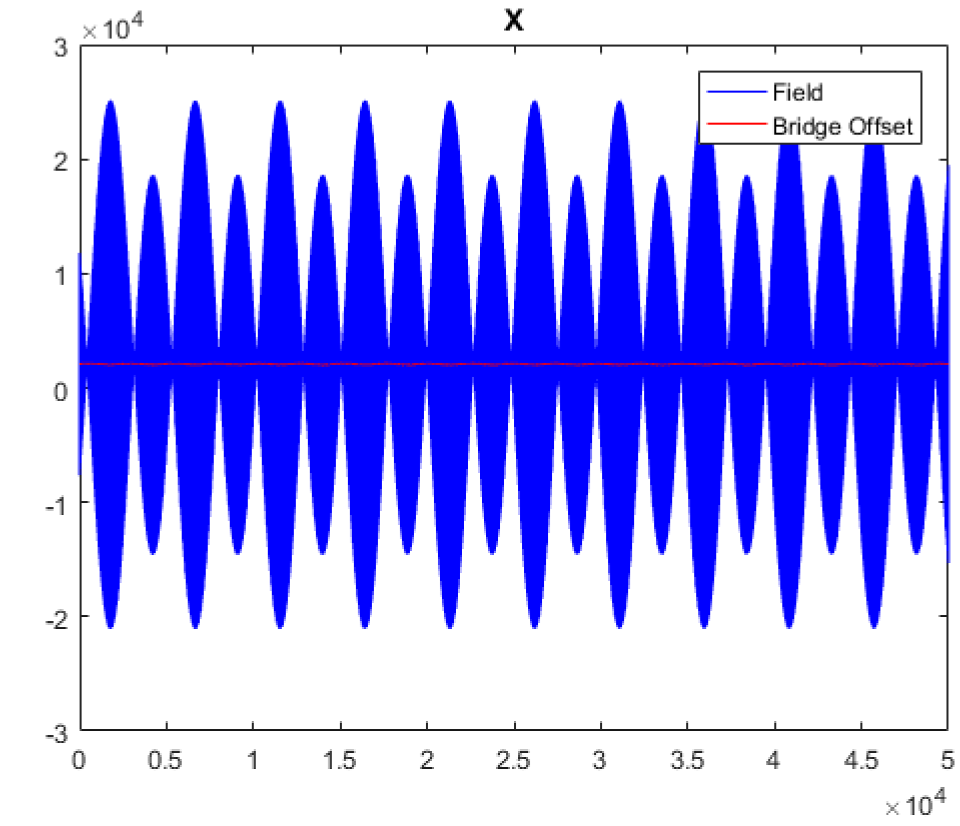


Figure 4.1: S17, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

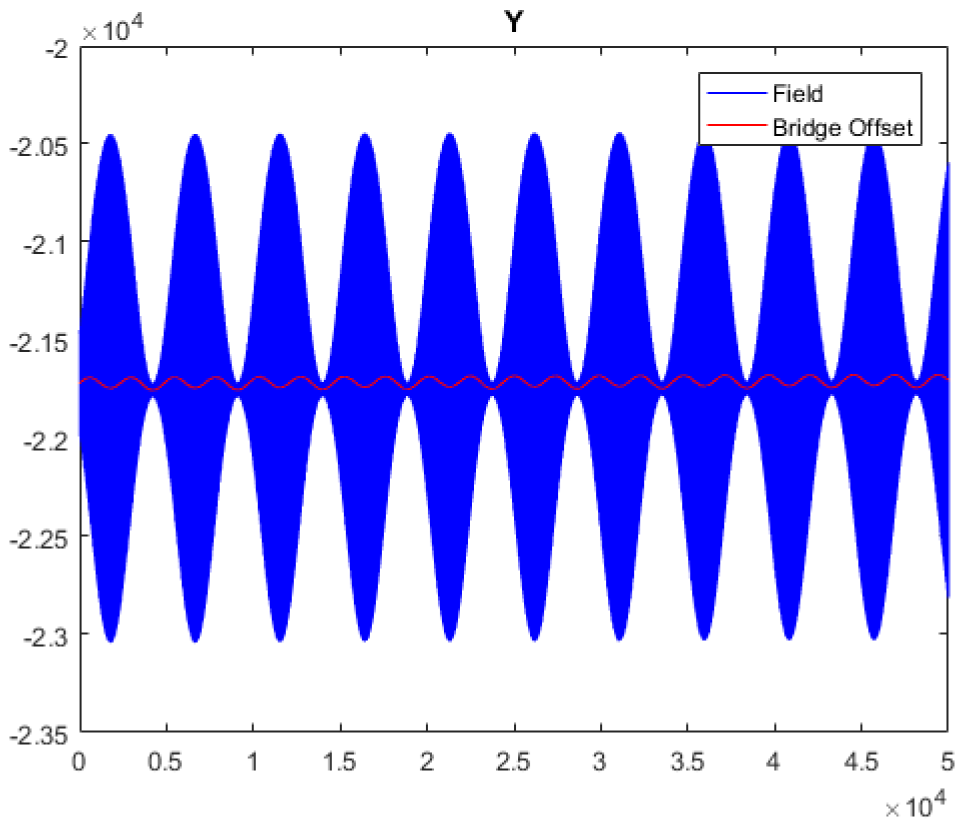


Figure 4.2: S17, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

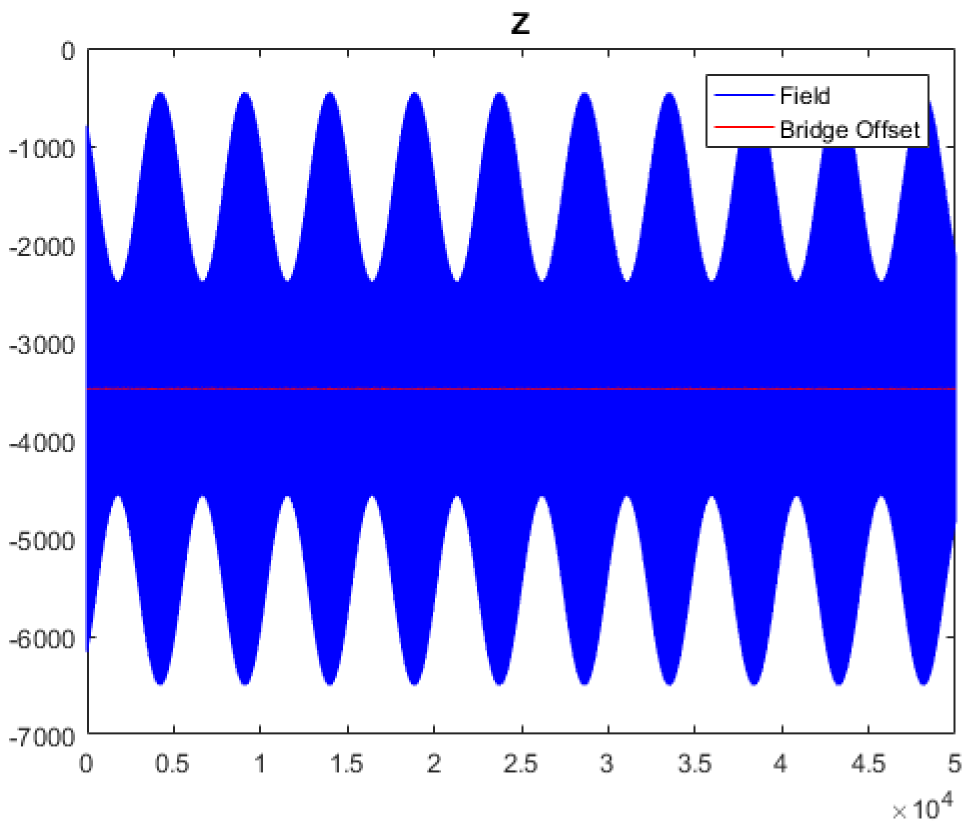
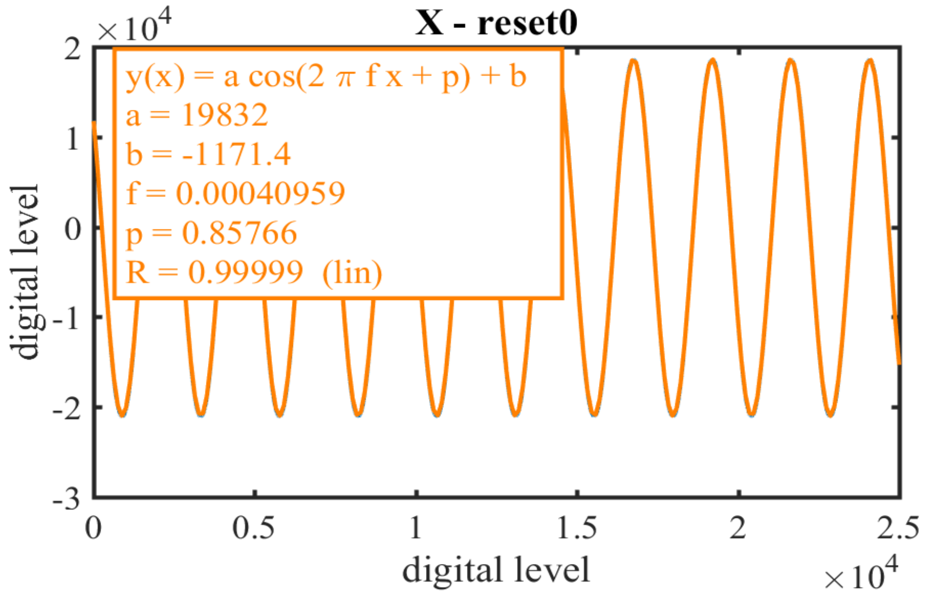


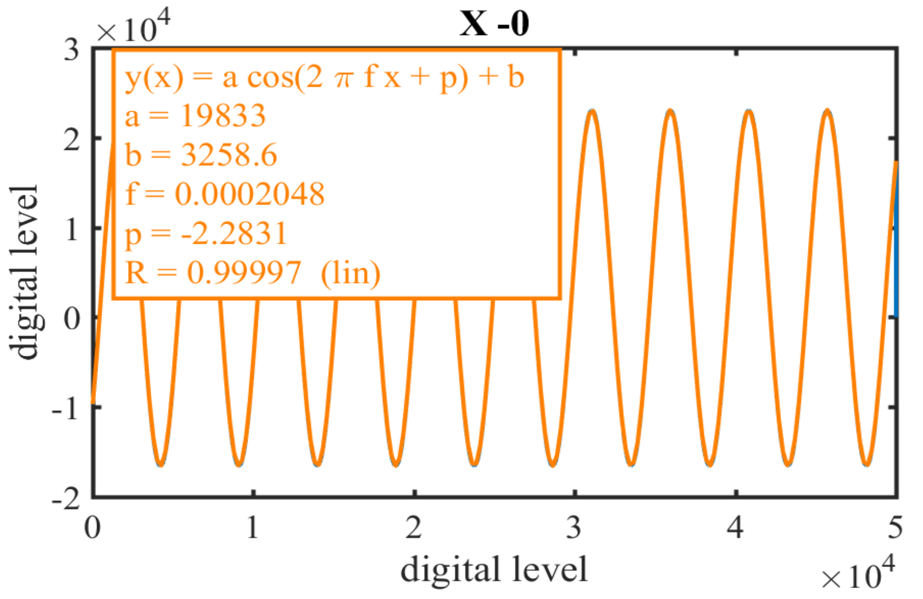
Figure 4.3: S17, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

Rest of the Results are in ***Appendix E.2***

As we can see, characteristics of all 18 sensors are not given. It is because some of them due to high bridge offset saturated even at extremely small applied magnetic field. Thus, they could be of no practical use unless the bridge offset is reduced.

But as we know the rest and reset pulses of both the sensors were not symmetric. To understand whether this affects their ability to measure the magnetic field, individual sensitivity of set and reset values were also measured. A script given in ***Appendix E.3*** was written for calculating the set, reset and the overall sensitivities the results of which are in ***Appendix E.4.***





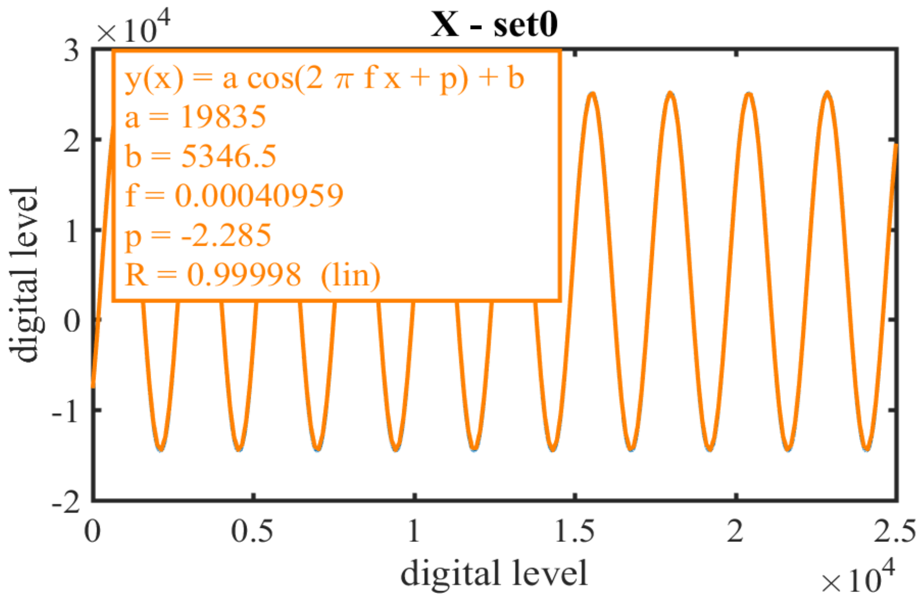


Figure 8.3: Amplitudes – Overall, Set, and Reset for data00000.txt

Rest of the Results are in ***Appendix E.4***

As we can see, some sensors got saturated. Remember that every sensor's characteristics were measured just before they reached their saturation point. The fact that still some sensors reached saturation point means that they have inherently very high bridge offset. This renders these sensors useless within the operation range to which the shall be subjected to, in future. We can also see that the regression of the curve fitting is not high for the sensors which got saturated. For the rest, we have fairly high linear regression. Therefore, to study the set and reset characteristics, we must restrict ourselves to the sensors where the regression is high. Consider the following results.

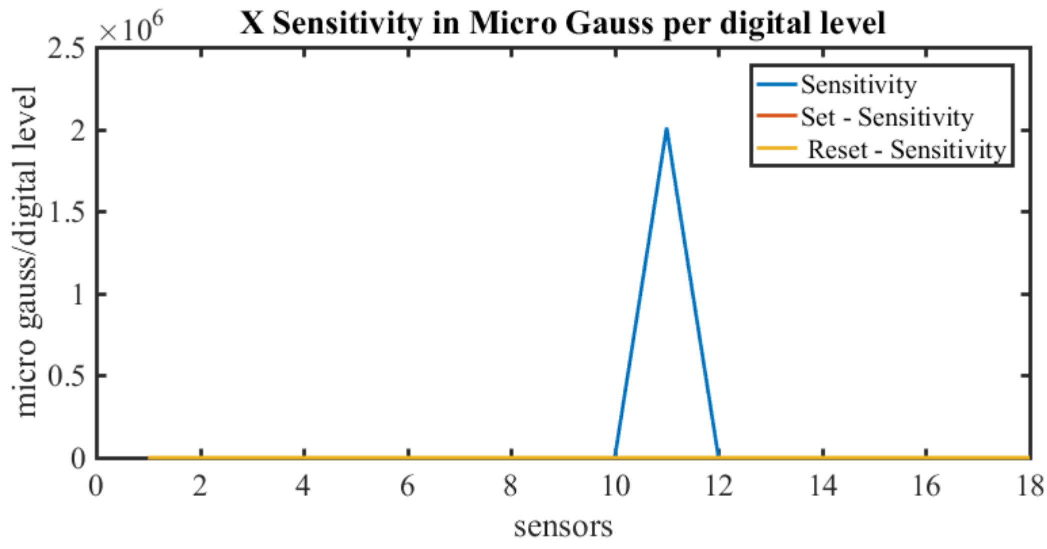


Figure 9.7: The set, reset, and overall sensitivity of all 14 sensors along x direction

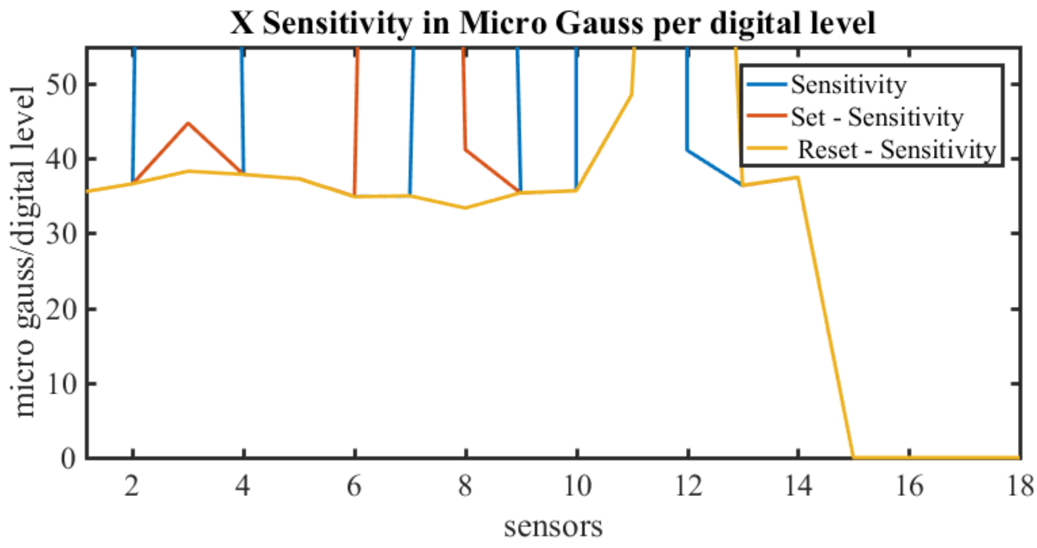


Figure 9.8: The set, reset, and overall sensitivity of all 14 sensors along x direction zoomed in on region of higher linear regression where we can see how they overlap perfectly

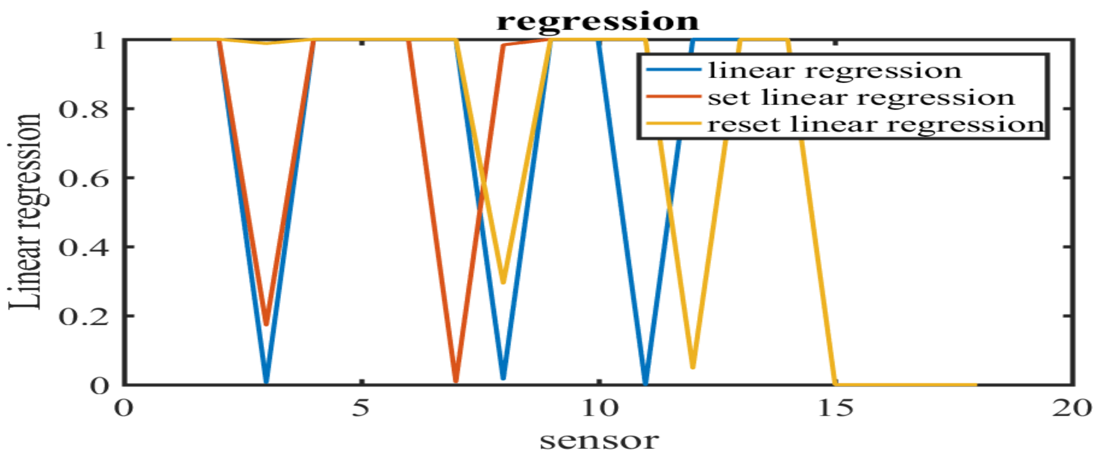


Figure 9.9: The set, reset, and overall linear regression of curve fitting for all 14 sensors

As we can see from f*igure 9.7, 9.8 and 9.9*, places where linear regression of the curve fitting is high, there is also a high degree of superimposition between the set, the reset, and the overall sensitivity suggesting that even though the set and reset pulses might not be identical, their difference is not high enough to cause trouble in the functioning of the sensor. The dependency of the amplitude of the measured signal with respect to the applied voltage by the function generator was also measured. See *figure 10.0*. Here the amplitude is expressed in digital level / 10^4 to make it possible to represent it on the same scale as the applied voltage which is in volts. As we can see, except for the saturation points, the dependency is totally direct, i.e. a straight line.

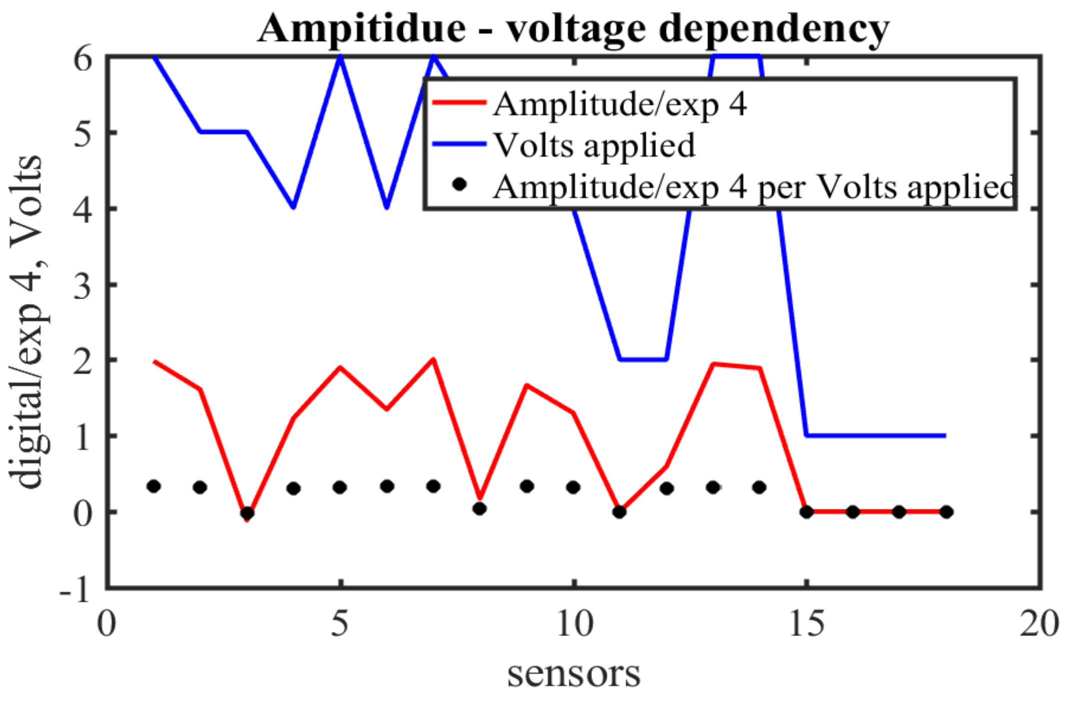


Figure 10.0: The amplitude – voltage dependency for all 14 sensors

The mean sensitivity was observed to be 35.97 µG/digital level.

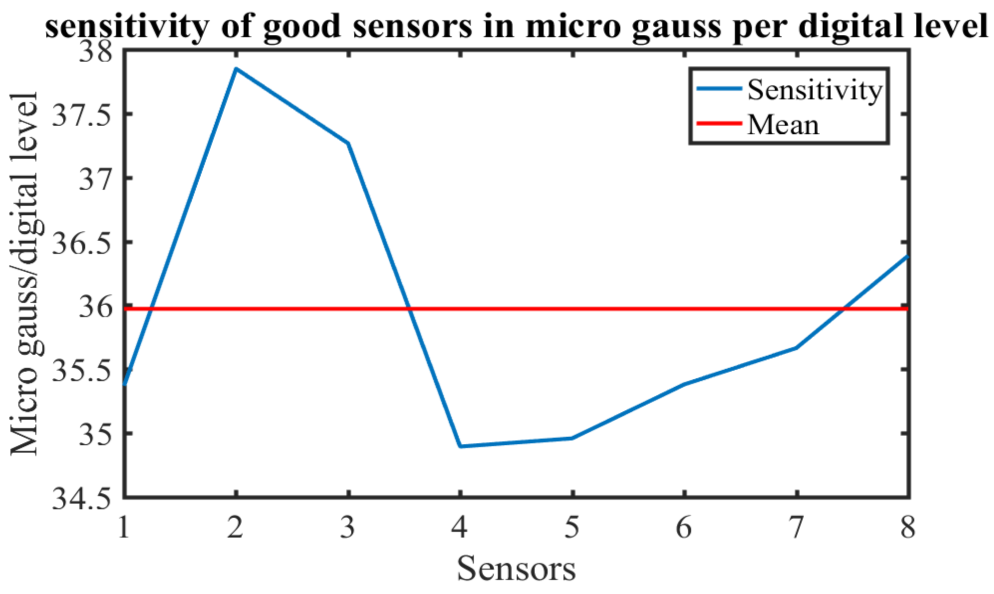


Figure 10.1: The sensitivity of all 8 no saturated sensors and their mean value

OBJECTIVE 5 | TO DETERMINE THE NOISE |

Noise is nothing but data which we did not intend to measure. It can be both electrical and mechanical and the source could be anything from electrical appliances to the components of the sensor itself. Measuring noise is important because it gives us a threshold level of magnetic field below which if we try to measure we would be overwhelmed by the noise. Also, it must be kept in mind that no matter the nature and source of a noise, it would always be perceived as a magnetic field in the raw data. Therefore, the true nature and characteristics of the noise cannot be reconstructed with absolution. However, we need not reconstruct the noise, we just need to identify it and measure the density.

CONTRIBUTION |

To measure the noise, first, we should start by measuring the electrical noise due to the sensor's components itself like the amplifier, the resistance, etc. This is because it could be isolated. The electrical noise could be measured separately by putting the sensors inside the zero gauss chamber which would shield the sensors from any magnetic noise. This way, when we take the measurement of the magnetic noise within the lab (which is a combination of both – actual magnetic noise and electrical noise), we can compare it with the electrical noise of the sensor's component to determine what is the actual magnetic noise. To do so, the sensors were put individually inside the zero gauss chamber and were made to take 10^6 samples (~1 hour).

Once the data from all the sensors were obtained, before processing any of it, it first must be properly partitioned. The partitioning is done to achieve two objectives:

1. Easily process the data
2. To account for the loss of data due to the way the amplitude is calculated.

The first point is self-explanatory but the second point needs more elaboration. We must consider how the actual value of the amplitude is obtained. Consider *figure 10.2*

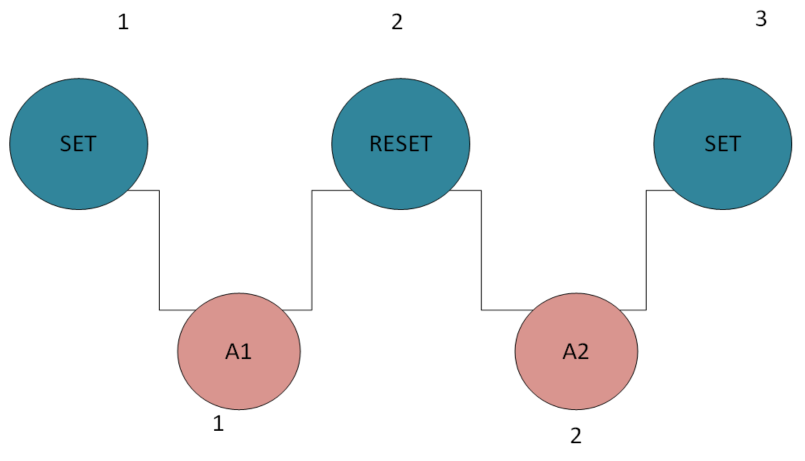


Figure 10.2: Problem with the algorithm for calculating amplitude from raw data

The amplitude is calculated as follows:

```
kk=1;
 for ii = 1:(n-1)/2 %n is the total number of points in the raw data
    ii1=ii-1;
       Vamp(kk) = (data(2*ii+1)-data(2*ii))/2; %Vamp is the amplitude
      k=k+1;
       Vamp(kk) = (data(2*ii1+1)-data(2*ii))/2;
      k=k+1;
 end
```

To describe it simply,

$$\text{Amplitude} = \frac{1}{2}\ (\text{Set} - \text{reset}) \quad \text{or,} \quad \frac{1}{2}\ (\text{Reset} - \text{Set})$$

Therefore, from 3 points in the raw data, we can only extract 2 points of amplitude, thus, for n points of raw data, we can only get n-1 points of amplitude. For 10^6 samples, we can only get 99999 values of amplitude. This is a problem since partitioning odd numbers is always a problem. The way to get around this problem is only by rejecting some extra points to make all the partitioned data have the same number of the element. Here, we reject 19 more points from the raw data to create 20 equal samples, each with a size of 49999. Given in ***Appendix F.1*** is the code used for partitioning the data.

Once the data was partitioned, the noise density and the RMS noise was calculated for each partition and was averaged over all the partitions. This was done for all the sensors. Given in ***Appendix F.2*** is the program written for the noise calculation and in ***Appendix F.3*** its corresponding output.

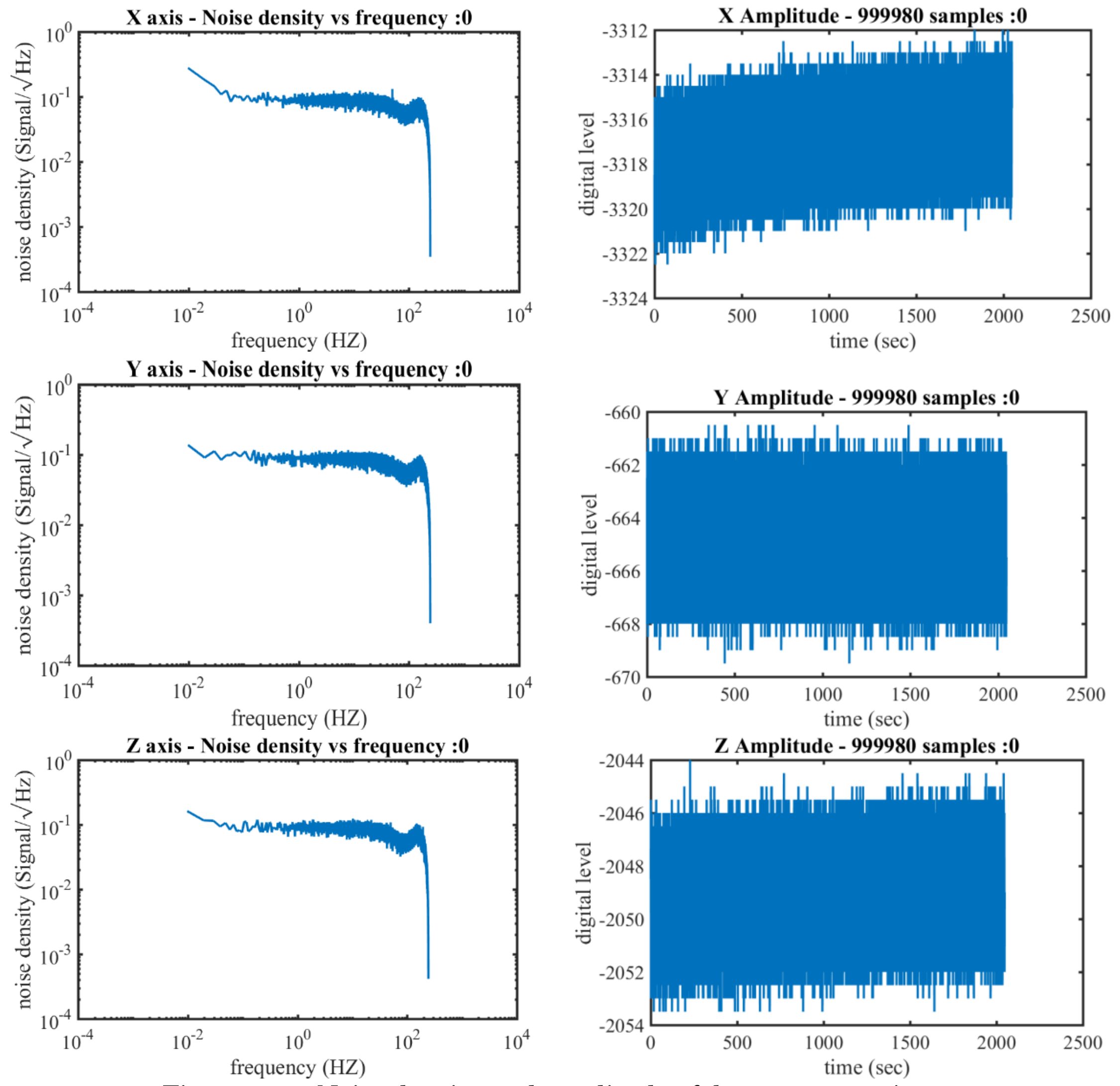


Figure 10.3: Noise density and amplitude of data00000.txt, i.e sensor s17

Rest of the Results are in ***Appendix F.3***

The values of average noise densities are as follows:

Sensor name	Average Noise Density-X axis (Signal/√Hz)	Bridge offset X (Signal)	Average Noise Density-Y axis (Signal/√Hz)	Bridge offset Y (Signal)	Average Noise Density-Z axis (Signal/√Hz)	Bridge offset Z (Signal)
S17	0.0850	2083	0.0840	-27150	0.0853	-3476
S7	0.4134	-10890	0.1254	-16150	0.2998	3747
S5	0.2854	-20510(sat)	0.1268	-13220	0.1079	-2340
S14	0.0864	-16170	0.0858	-11500	0.0860	-605.5
S10	0.0861	-3529	0.0865	-26690	0.0863	-19130
S6	0.0834	-14050	0.0818	-5724	0.0830	-11580
S8	0.0808	-10630	0.0818	-17680	0.0818	-3699
S1	0.0844	-21920(sat)	0.0860	-28510	0.0881	384
S16	0.0840	-10580(sat)	0.0831	-19480	0.0853	-325.5
S2	0.0840	-16760	0.0833	-9548	0.0833	-658
S12	0.0842	-2832(variant)	0.0889	-23580	0.0903	-646.5
S3	0.0818	-20650	0.0838	-24570	0.0833	-9745
S15	0.0835	-5088	0.0836	-8031	0.0850	-10140
S13	0.3904	-2530	0.1196	-19220	0.3173	2964

Before we tabulate the values of the RMS noise, we must quickly go over what we mean by the RMS noise. When we talk about the RMS noise, we actually mean the noise around a particular signal-which is the mean. The actual RMS noise would give the overall noise including the mean value which is some offset which can be easily compensated for. Therefore, in a sense, the actually RMS is not of much use since it tells little about the sensor itself. On the other hand, when we talk about RMS noise around a signal, it is more useful since it excludes the DC offset and represents only the actual spread of the noise around the offset. Thus, to verify the RMS noise calculated in the frequency domain, we must correlate it with the standard deviation of the signal in the time domain rather than the RMS of the signal in time domain. Following are the values of the RMS and STD (σ) in frequency and time domain respectively.

Sensor name	RMS Noise X (Signal)	σ X (Signal)	RMS Noise Y (Signal)	σ Y (Signal)	RMS Noise Z (Signal)	σ Z (Signal)
S17	0.9917	1.1178	0.9784	0.9793	0.9933	1.0150
S7	4.5808	4.5813	1.4388	1.4389	3.2680	3.2700
S5	2.3368	2.3395	1.2624	1.2656	1.2272	1.2280
S14	1.0044	1.0141	0.9993	1.0458	1.0037	1.0189
S10	1.0015	1.0074	1.0049	1.0111	1.0029	1.0032
S6	0.9742	0.9819	0.9557	0.9565	0.9680	0.9697
S8	0.9457	0.9989	0.9547	1.0150	0.9526	0.9619
S1	0.9872	0.9874	1.0013	1.0025	1.0311	1.0341
S16	0.9801	0.9814	0.9668	0.9689	0.9953	0.9956
S2	0.9769	0.9769	0.9706	1.0354	0.9716	0.9751
S12	0.9795	0.9795	1.1007	1.1011	1.4444	1.4445
S3	0.9533	0.9538	0.9762	0.9803	0.9673	0.9683
S15	0.9733	0.9736	0.9750	0.9757	0.9905	0.9907
S13	4.6275	4.6281	1.4291	1.4313	3.5951	3.5956

Before we proceed, there is a critical thing to observe here. As we can see, the standard deviation agrees quite well with the RMS noise but not completely. This small discrepancy is important and it can be explained by considering how STD and RMS noise is being calculated. The RMS is calculated as,

$$Power[\,f\,]_i = \frac{1}{c}\ \sum_{j=1}^{c} abs(fft\ (x[\,t\,]_i - \mu_j\,)^2\,)/\,r^2$$

$$Power\{\,1{:}\,end\,\} = Power\{\,1{:}\,(r{+}1)/2\,\}$$

$$Power\{\,2:end\,\} = 2 \times Power\{\,2{:}\,end\,\}$$

$$Noise[\,f\,]_i = \sqrt{\frac{Power[\,f\,]_i}{\partial f}}$$

$$RMS = \sqrt{\sum_{i=1}^{\frac{r+1}{2}} Noise\ [\,f\,]_i^{\ 2}\ \partial f} \ = \ \sqrt{\sum_{i=1}^{\frac{r+1}{2}} Power\ [f]_i}$$

Whereas, the standard deviation is calculated as,

$$x\{\,r\,,c\,\} \ \rightarrow \ x\{\,r \times c\,,1\,\}$$

$$\sigma \ = \ \sqrt{\frac{1}{r*c-1}\sum_{i=1}^{r*c}(x[t]_i - \mu)^2}$$

The critical difference comes in the very first stage where in the case of RMS, we essentially find the standard deviation for each partition and then average over the total number of partitions whereas, in the case of standard deviation, we directly find out the standard deviation over the entire length. This causes the difference because the mean value for each partition might not be same. Now that we know the reason behind the small deviation between the RMS and the STD values, we can now move on as even after the discrepancy, they still show a very strong degree of correlation which shows that the values obtained are correct. Nonetheless, the values are still in terms of signal and also includes the values of the sensors with high bridge offsets which we have already established before that they are of not much use in their current state. So, A script given in ***Appendix F.4*** was written to measure and the express the RMS noise and the white noise level i.e. the noise density in terms of magnetic field. Given below are its results.

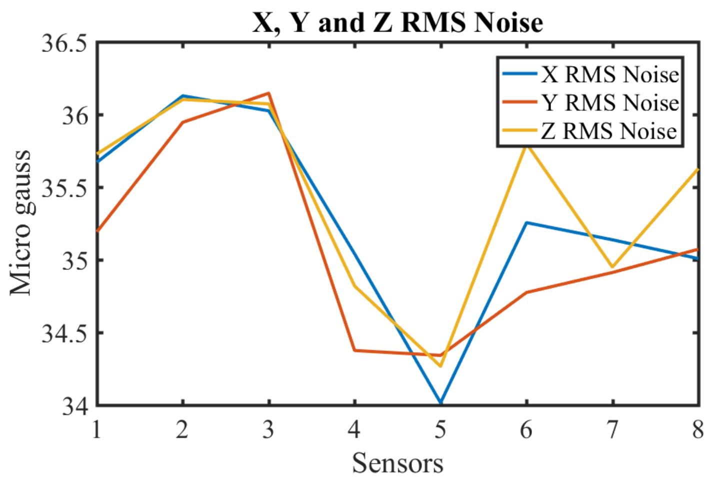


Figure 11.7: The RMS noise across all channels – X, Y, & Z for the eight best sensors

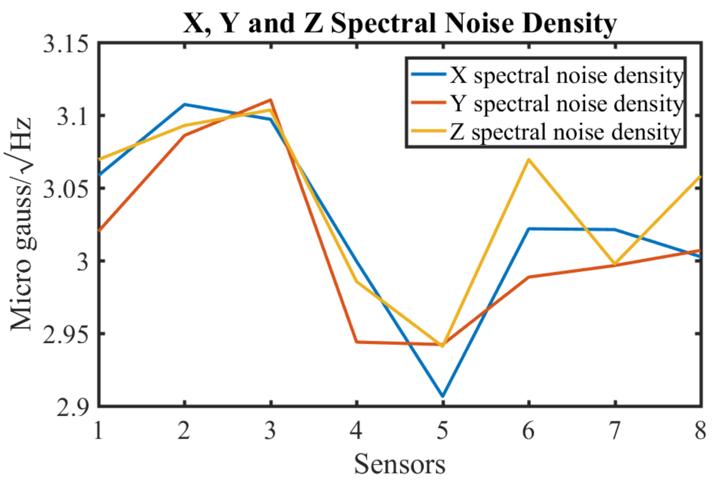
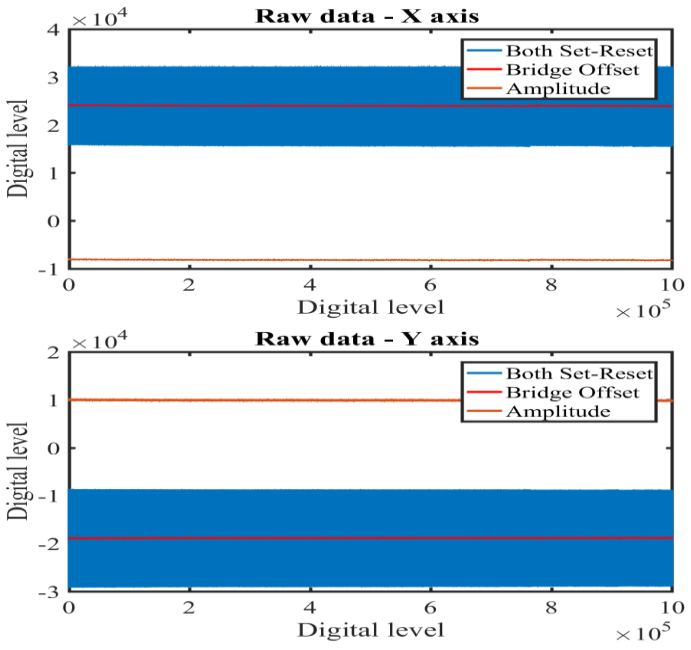


Figure 11.8: The Spectral Noise density across all channels – X, Y, & Z for the eight best sensors

The maximum RMS noise for all the sensors across all the channels was found to be 36.15 μG that is the maximum a signal can spread around the mean value is 36.15 μG which is nearly 100 times less than the value required to be measured by the sensor (3.57 mG), i.e. 1.0126 % deviation which is quite acceptable, and the maximum spectral noise density for the white noise for all the sensors across all the channels was found to be 3.11 μG/√Hz and the white noise bandwidth was found to be ≈ 36.5 Hz. Therefore, the lowest magnetic field one could measure using this sensor in a single reading without being overwhelmed by the noise would be 18.7891 μG which is nearly 200 times less than the value to be measured (3.57 mG), i.e. we can measure as low as 0.5263 % the required value which is quite nice.

Now that we have calculated the noise due to the electrical components of the sensor itself, we can now determine the magnetic noise of the lab. For this, we can measure the noise outside the ZGC and subtract from it the noise measured inside ZGC. The script is given in ***Appendix F.5*** and ***Appendix F.6*** was written to analyze the data and given below are its results.



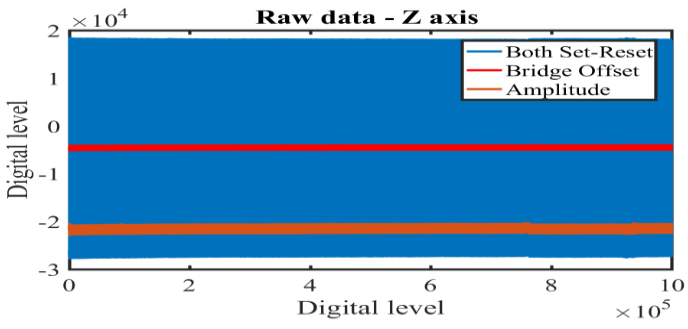


Figure 11.9: Readings for noise outside ZCG

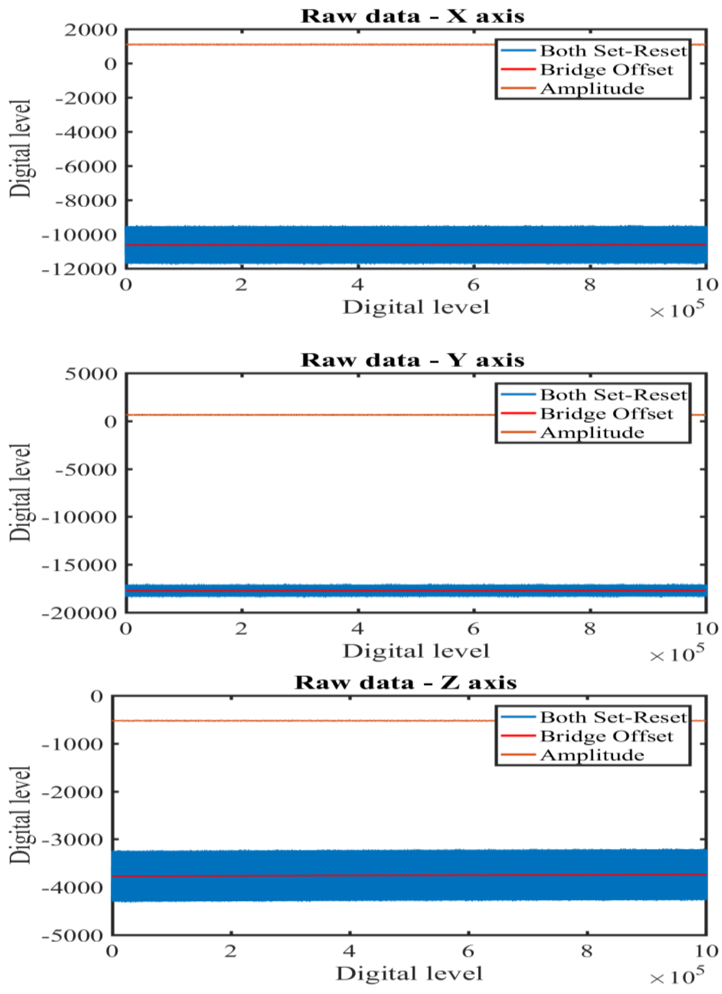


Figure 12: Readings for noise inside the ZGC

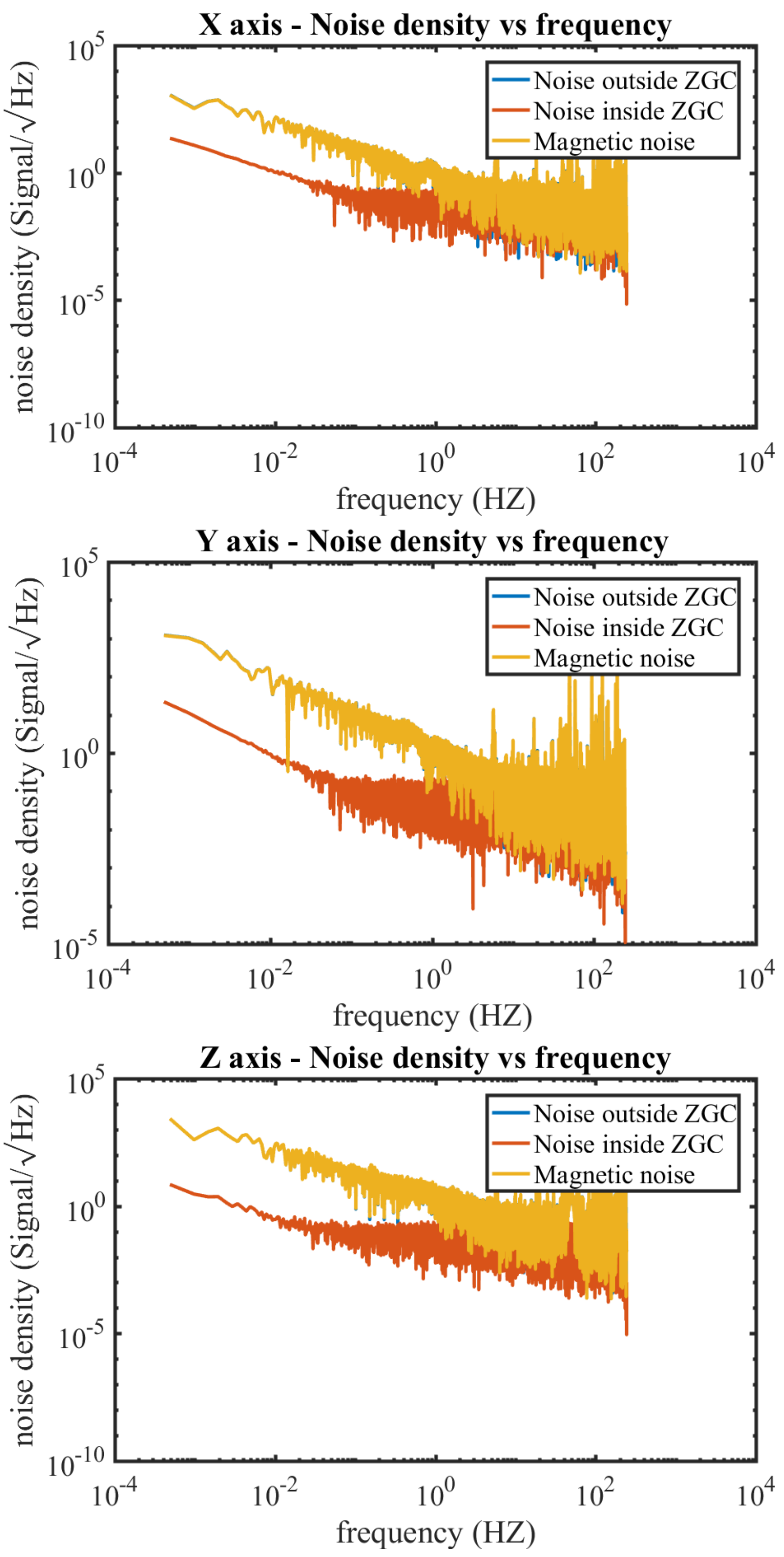


Figure 12.1: Net Noise (outside ZGC), Electrical Noise (inside ZGC) & Magnetic Noise.

As we can see, the electrical noise contributes very less to the overall noise as the magnetic noise almost superimposes with the net noise thereby showing the contribution of the electrical noise to the net noise is negligible. If we measure the RMS noise for the three cases, we will get the following results:

Along X axis:

Overall RMS Noise:	64.7326 signal
RMS Noise due to the stray magnetic field:	64.4116 signal
RMS Noise due to electrical noise:	01.1464 signal

Along Y axis:

Overall RMS Noise:	118.6724 signal
RMS Noise due to the stray magnetic field:	118.4620 signal
RMS Noise due to electrical noise:	001.1342 signal

Along Z axis:

Overall RMS Noise:	671.8988 signal
RMS Noise due to the stray magnetic field:	671.9110 signal
RMS Noise due to electrical noise:	000.9720 signal

Therefore, we can successfully say that the overall measured noise is very close to the actual value of the noise due to stray magnetic fields in the lab.

OBJECTIVE 6 | TO DETERMINE THE ROBUSTNESS |

Once all the sensitivity and the noise were measured, now it was time to determine the robustness of the sensors. There are mainly three parameters that fall under robustness:

- How consistently the interface program recognizes all the sensors after switching the power on.
- How long and how consistently can the sensors continue to transmit data without encountering some fault.
- How consistent are the data under similar circumstances

CONTRIBUTION |

Initially, the interface program was found to be quite robust recognizing all the sensors nearly 8 out of 10 times. Then the consistency of the sensors was tested. They were made to record data continuously for 4 hrs, 12 hrs, and 24 hrs. before stating the conclusion, a point worth mentioning is the presence of “spikes” in the raw data. This was something which plagued and lowered the pace of the experimentation for several days. A spike is when the magnitude of the measured field suddenly alters by a drastic amount and the change is restored within few milliseconds. There were mainly two classes of spikes that were encountered.

Class 1 Spike :

In a class 1 spike, the results would suddenly jump due to saturation. Corresponding to the spike, the values for each channel would reach values beyond 32000, and the data obtained would look like *figure 12.2 – Zone A*. When a sensor demonstrates this class of spikes, it means that the bridge offset of that particular sensor is very high. Therefore, the only effective way to permanently deal with it is to apply shunt resistance on the bridge and lower the offset.

Class 2 Spike :

This is a very critical and complex kind of spike. In this class of spikes, the problem would be a compound effect of two phenomena occurring successively:

1. The counter would lag and then skip a count
2. The set (or the reset value) would suddenly change to match the value of reset (or set).

The following example shall elaborate the above-stated points. Given below is an example from the actual raw data code for sensor B5-X channel.

-6011	104
12340	105
-12360	105
-6011	107

This type of spikes happened in our case because of improper wiring but it can also happen if the critical phenomenon of "de-synchronization" - when the synchronization is lost due to long strings of 0s or 1s is successively followed by the "failed set-reset pulse" – where a particular set reset pulse fails to reach the threshold trigger level. This class of spike is highlighted in *figure 12.2 – Zone B.*

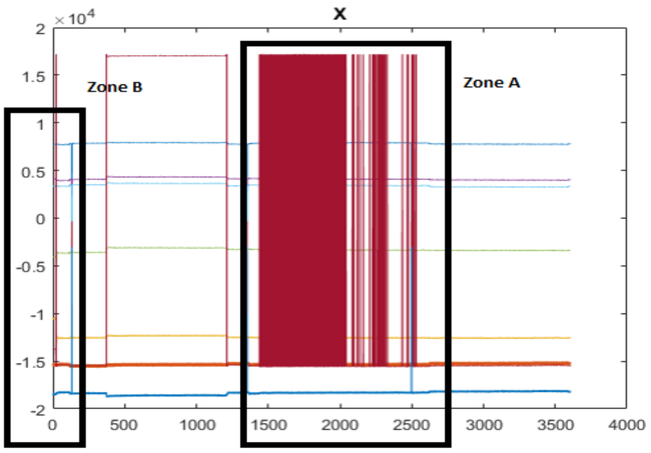


Figure 12.2: Class 1 and Class 2 spikes.

It is also worth mentioning that fluctuations in power supplied will also cause spike, therefore, it is recommended that the negative terminal should also be grounded.

Once the wirings causing the spikes were rectified, sensors were then found to be quite robust as they successfully managed to take nearly ten million samples for 24 hours without showing any spikes or inconsistencies. At this point, long cables were being made to be able to test the robustness of the sensors over longer connections. Here unexpectedly, a very critical issue was encountered. For some reason, the CAN adapter seems to recognize only 3 sensors (B sensor 1, B sensor 3 and B sensor 7) when connected to all the 8 sensors. However, as soon as any one of the sensors were disconnected, the CAN adapter immediately recognized the remaining 7 sensors. This was found to be valid no matter which sensor was disconnected. To understand the issue, can high and set reset signals were studied for various cases.

- Case 1 : Long cable, 7 sensors
- Case 2 : Long cable individual sensors
- Case 3 : Short cable, 8 sensors
- Case 4 : Short cable, 7 sensors
- Case 5 : Short cable, individual sensors

The preliminary results of individual sensors from cases 1 to 5 are given in ***Appendix G.1 to G.5.*** The main results from cases 1 to 5 are given below:

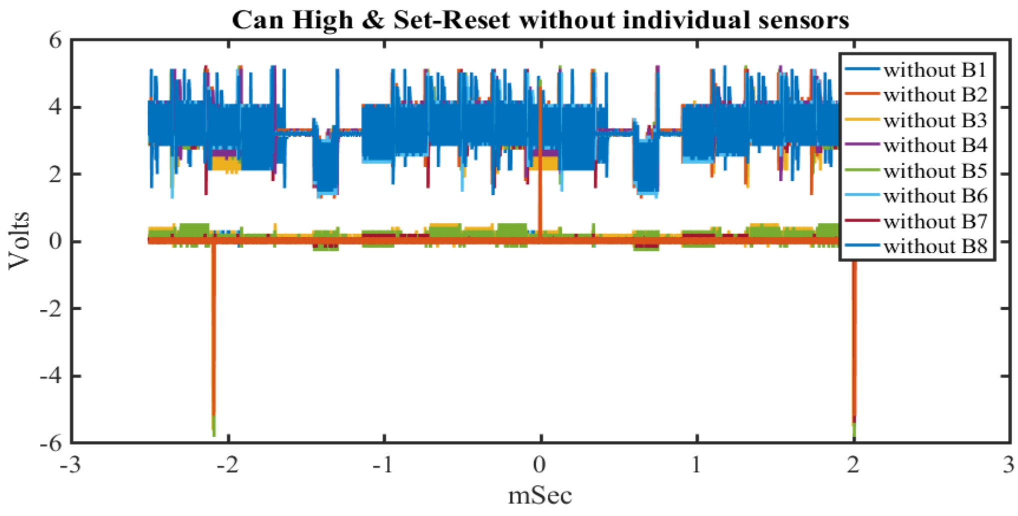


Figure 14.9 : Combined result from case 1

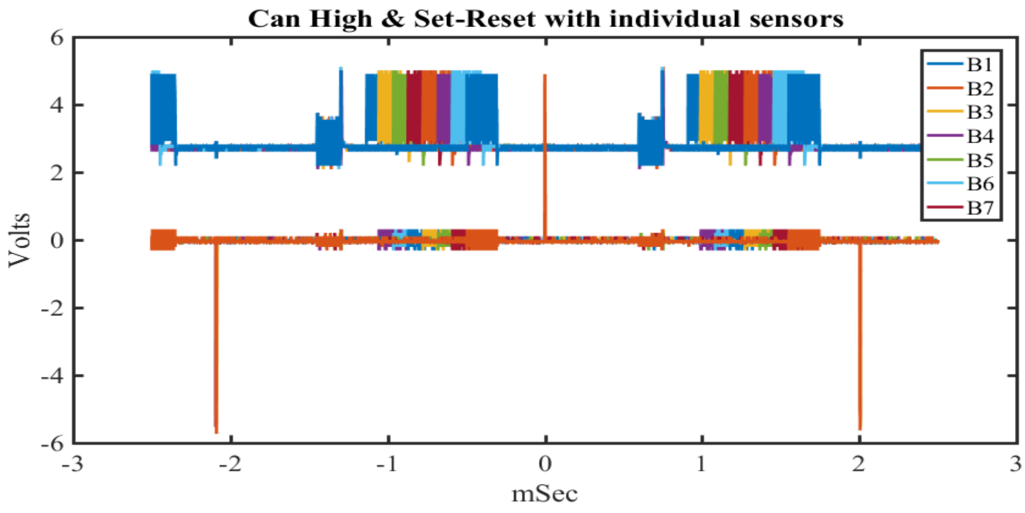


Figure 15 : Combined result from case 2

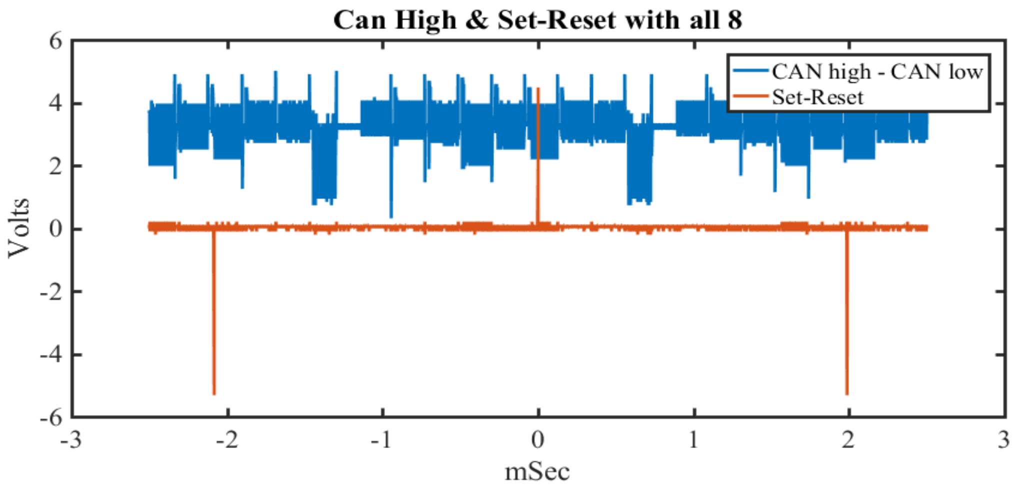


Figure 15.1 : Combined result from case 3

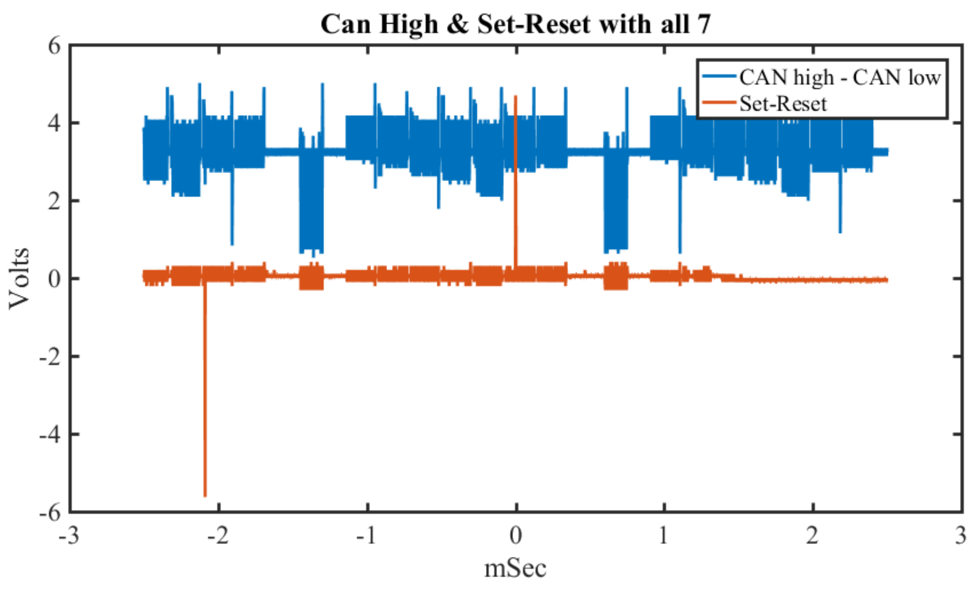


Figure 15.2 : Combined result from case 4

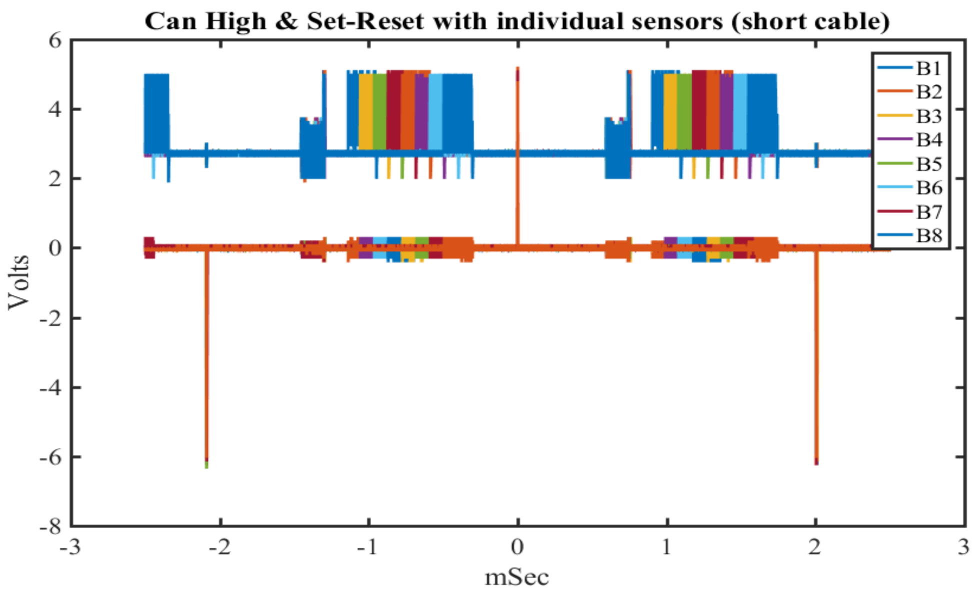


Figure 15.3 : Combined result from case 5

- From cases 1 and 2, we can conclude that the counter that dictates which sensor sends data, arrives at the respective values for the sensors much before the sensors are able to process and send the data. Thus, individually, they finish processing and sending data much faster. On calculating, we find delay associated with each sensor when working together using the long cable is ≈ 0.1352 mSec.
- From cases 3 and 5, like cases 1 and 2, it was found that in the case of the short cable also, there was a delay associated with the sensors when they are made to work together. The delay associated with each sensor was calculated to be ≈ 0.1256 mSec.
- From cases 2 and 5, we find that the time taken by an individual sensor to send the data is 0.205 mSec for short cable and 0.235 mSec for a long cable.
- However, careful examination shows that these delays associated are not constant. This poses a problem further deduction from the above-mentioned data was not going to be conclusive. Therefore, further investigation was needed.

Another thing which could be conclusive is the position of the ADC, where the conversion happens. In principle, this should happen at the same time for all the sensors without any perceivable delay. Thus, the ADC position was measured next.

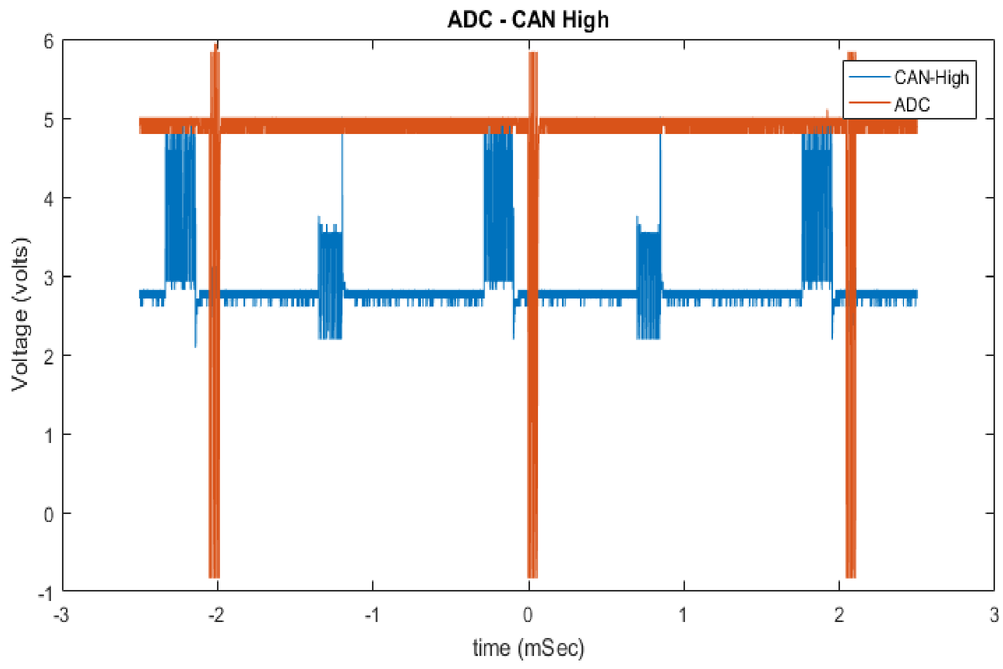


Figure 15.4 : ADC vs CAN High

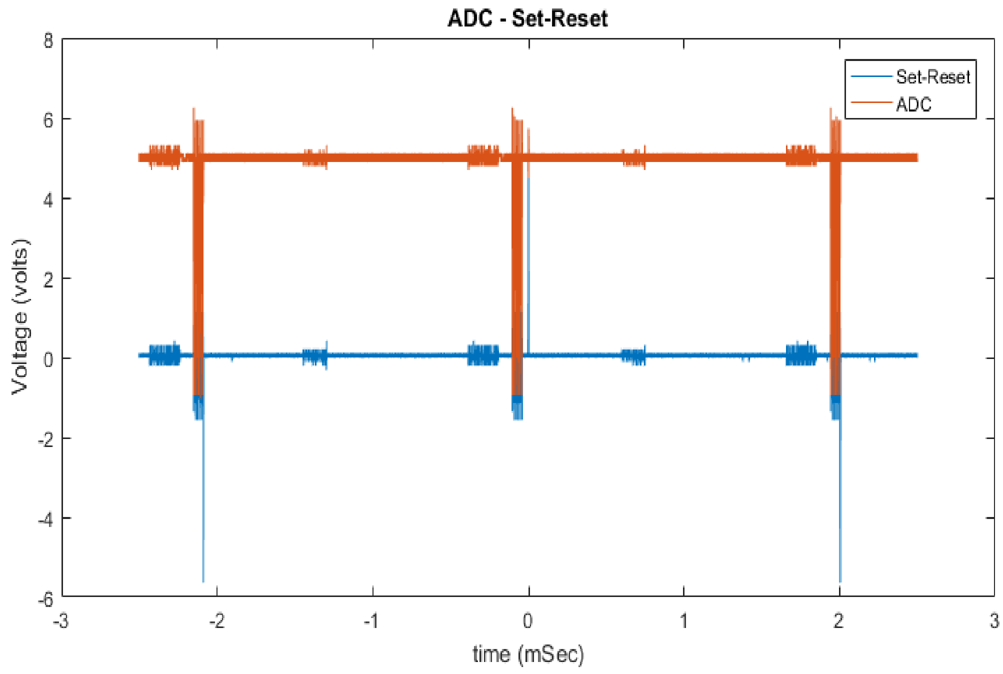


Figure 15.5 : ADC vs Set-Reset

Given in *figure 15.4 and 15.5* are the relationship between ADC, can and Set-Reset. The following results can be combined using the measured empirical relationship:

$$t_{set\text{-}reset} = t_{ADC} - 0.1014 \pm 0.001 \quad (\textit{all in mSec})$$

The result of this temporal shift is given below in *figure 15.6. figure 15.7* shows the ± 0.001 tolerance.

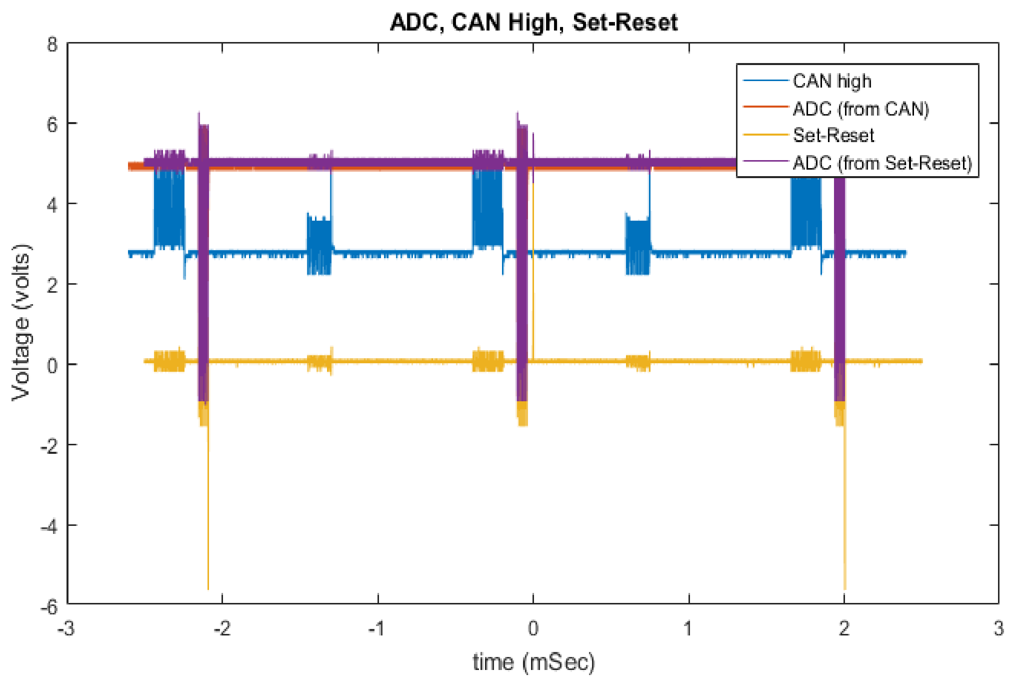


Figure 15.6 : ADC, CAN High, and Set-Reset

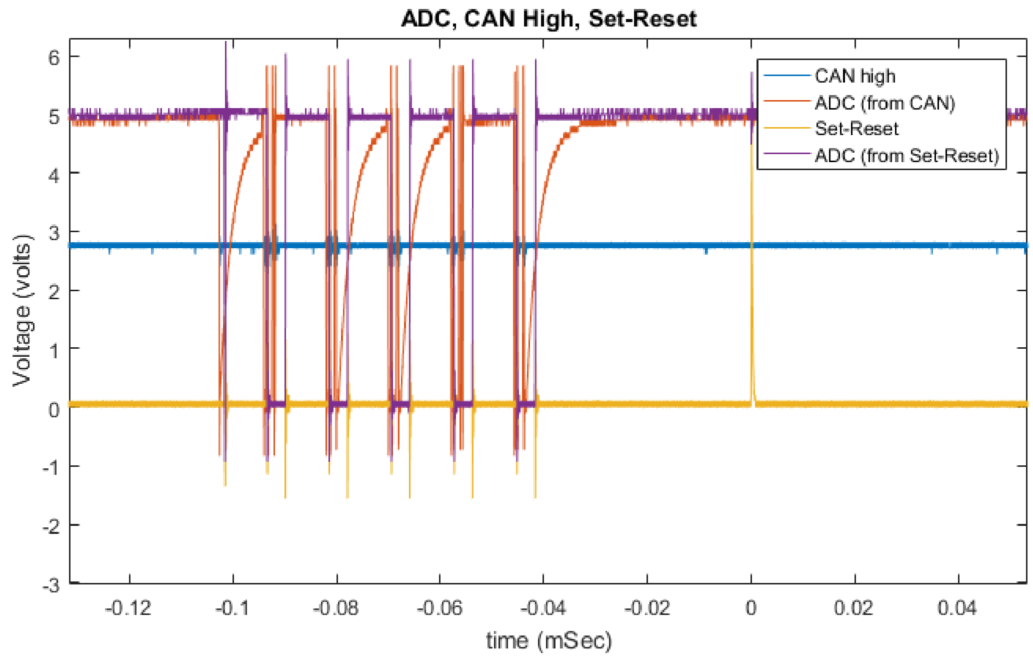


Figure 15.7 : ADC, CAN High, and Set-Reset (Zoomed in)

From figure *15.4, 15.5, 15.6 and 15.7*, the conclusion was drawn that the ADC conversion occurs first followed by the Set-Reset and then the uploading of data into the buffer by the sensor. The most important part of this entire process is the beginning of the counter. The program uses a prescaler of 128 and the sensor operates at 16 MHz with the ADC happening at the count of 67. It is also worth noting that the sensor itself operates with a prescaler of 256 thus it should be checked where the counter is and what prescaler is being used. The effective frequency should follow the relation:

$$f_{eff} = \frac{f_{I/O}}{N(1 + ADC\ Counter)}$$

Shown in *figure 15.8* below are the three points from where the counter should begin depending upon the choice of prescaler. Upon examination, it was eminent that the choice of prescaler being used for ADC was 256.

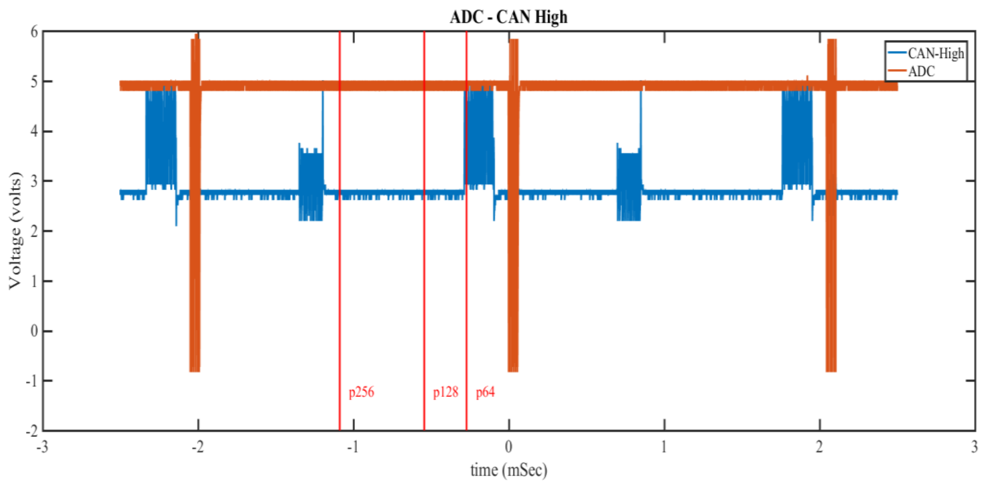


Figure 15.8 : ADC, CAN High, and Set-Reset (Zoomed in)

There is yet another prescaler that is being used. This one is for the data position. To find out its value, the following relation was derived and implemented.

$P(C+1) = K_f T$

Where $K_f = 16\times10^6$, P is the prescaler value, C is the counter value and T is the time taken by the counter to reach the counter value. From the above relation, we get,

$K_f T_n \ = n\, K_f T_0$

$K_f(t_n - t_o) \ = n\, K_f(T_0)$

$T_0 \ = (t_n - t_o)/n$

$P(c_0 \ + 1) = \ K_f(To)$

Values of t_n , t_o were found to be 1.137 and 1.0560 ms for n = 1500. Therefore, the value of T_0 and thereby, P was found to be = 54 ns and 0.864 respectively. Thus, the theoretical value of prescaling should be 1, that is no prescaling is used for the data position counter. This gives us essential insight into the minimum separation in data position of 2 consecutive sensors. At this point we return to experimentation with the short cable since we can manipulate all 8 of them at once with this cable.

As measured previously, the width of each signal pulse by a sensor is about 200.5 μs using which, we get a minimum separation of 3713 points for the short cable. Subsequently, for the long cable with a pulse width of 235 μs, the minimum separation in data position should be 4352 points.

At this point, another cable – ethernet cat 6, 3m long. This gave an insight regarding another potential problem – the relative position of the sensors. For some reason, the cable detected and measured with all 8 sensors only when they were connected in the weird sequence of their device ID. Now it must be inspected whether additional delay when connected non-sequentially is enough to corrupt a signal. Given in ***Appendix G.6*** are the values of all the sensors when measured connected via the ethernet cable with minimum spacing factored in. Given below in *figure 16.7* is a superposition of those figures showing that the spacing was indeed enough to separate them.

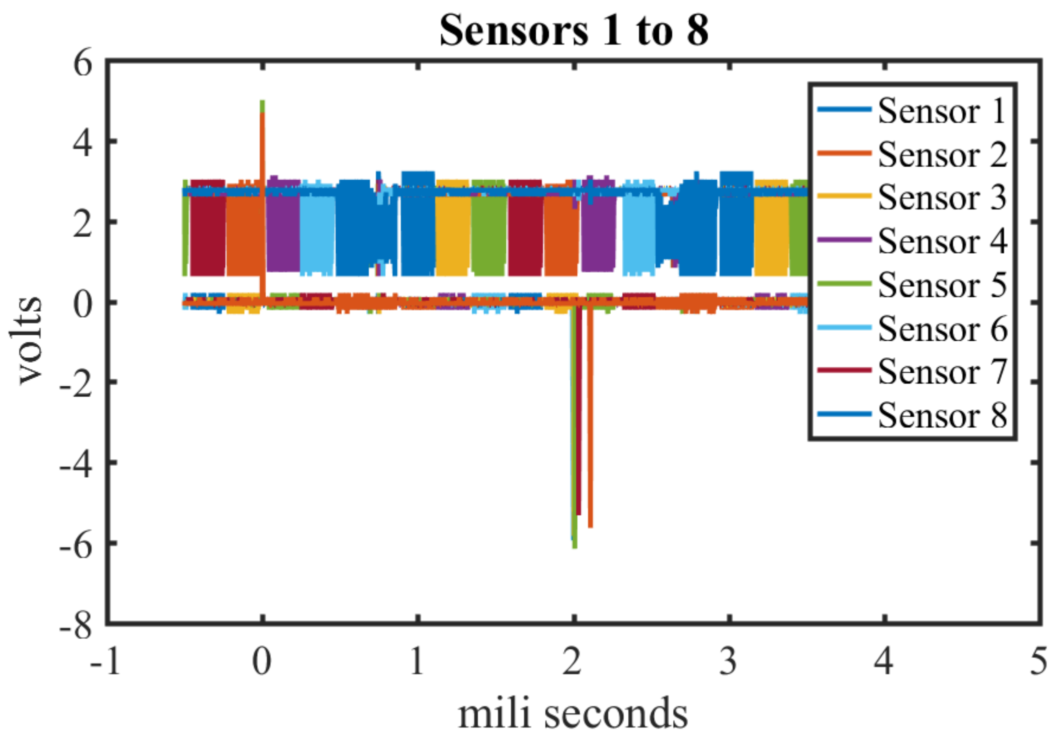


Figure 16.7 : ADC and CAN High of all sensors as measured using the ethernet cable

It was finally found that the problem was due to terminating resistors. Terminating resistors are impedances used to order to minimize the voltage losses caused by the effective resistance of the cable, whose value for long cables are prescribed to be $118\,\Omega < R_T < 150\,\Omega$. The terminating registers used in our case is $120\,\Omega$. The problem was that each resistor had its own terminating resistor while in principle, it should be only the last one in the bus, who has the terminating resistor. Removing terminating resistors from the remaining 7, (only B sensor 1 has it) immediately solved all the robustness and detection issue. It was tested with both, a shielded twisted pair cable and a cat – 7 ethernet cable each of them 10 m long and they showed remarkable robustness, running for days without stopping and immediate detection when rescanned. *Figure 16.8* demonstrates the proper use of terminator.

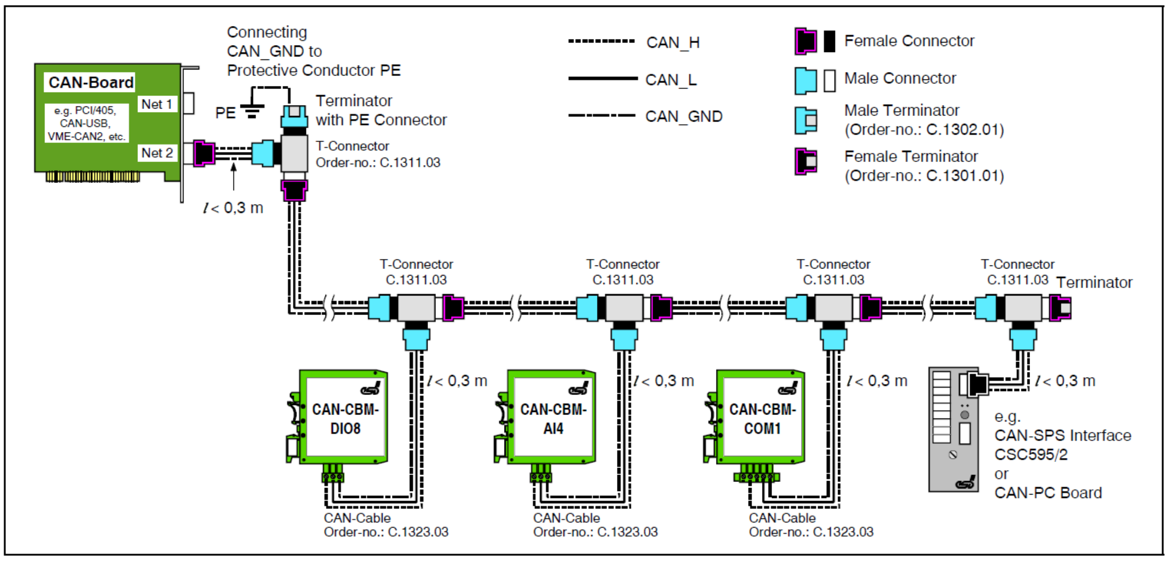

Connecting CAN_GND to Protective Conductor PE
CAN-Board
e.g. PCI/405, CAN-USB, VME-CAN2, etc.
Net 1
Net 2
PE
Terminator with PE Connector
T-Connector Order-no.: C.1311.03
l < 0,3 m
CAN_H
CAN_L
CAN_GND
Female Connector
Male Connector
Male Terminator (Order-no.: C.1302.01)
Female Terminator (Order-no.: C.1301.01)
T-Connector C.1311.03
Terminator
CAN-CBM-DIO8
CAN-CBM-AI4
CAN-CBM-COM1
CAN-Cable Order-no.: C.1323.03
e.g. CAN-SPS Interface CSC595/2 or CAN-PC Board

Figure 16.8: A figure from esd gmbh Hannover CAN-Wiring Notes on how to use terminator

Finally, the sensors were mounted on the actual compensating coils and were made to measure the data while being subjected to the strong magnetic field of the coils. While previous sensors would shut down and not read until restarted, these new ones not only not shut down but also continues to measure even when subjected to the strong alternating field of the coil. This shows that they are indeed, much more robust than the previous ones.

Figures 16.9, 17.0 and 17.1 shows how the sensors resume their measurements even when being subjected to the alternating magnetic field of the coil.

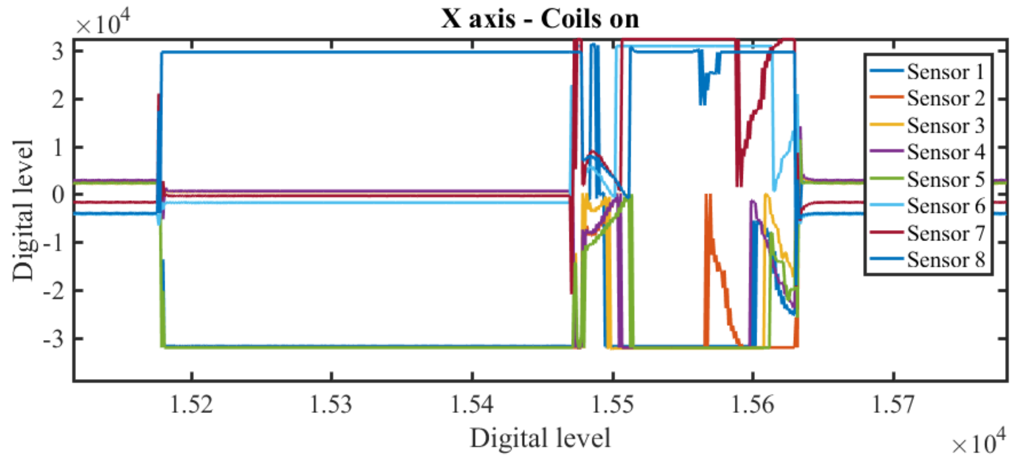


Figure 16.9 : Response of all 8 sensors along X axis when subjected to high magnetic field

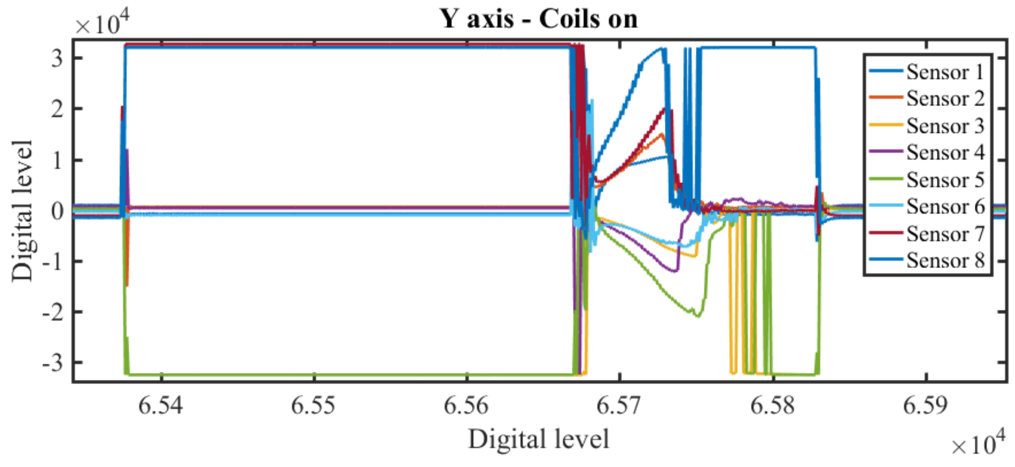


Figure 17 : Response of all 8 sensors along Y axis when subjected to high magnetic field

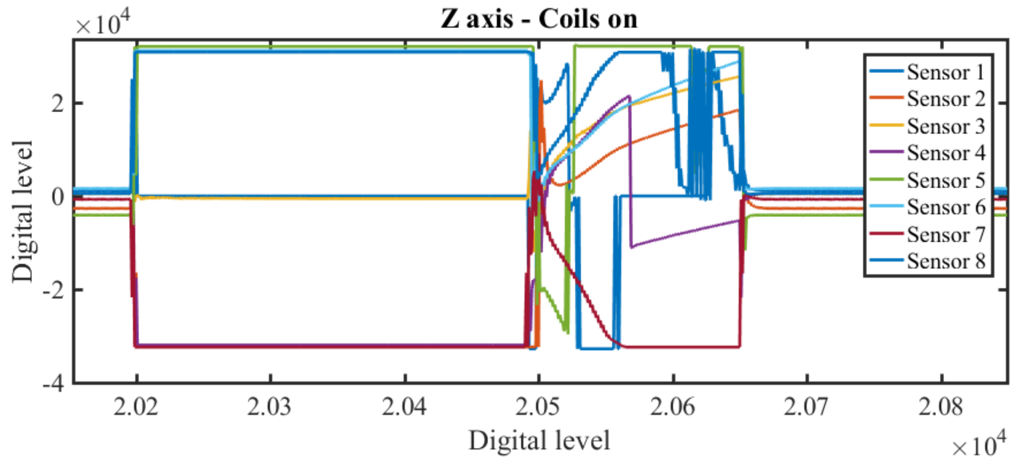


Figure 17.1 : Response of all 8 sensors along Z axis when subjected to high magnetic field

OBJECTIVE 7 | FINAL CALIBRATION |

CONTRIBUTION |

In final calibration, the sensors were pushed to their limits and were optimized to the maximum threshold. The following optimizations were achieved:

1. The timing of the ADC occurrence was optimized. Previously it occurred between the fourth and the fifth sensor there by posing a consistency problem. It meant that for every data set, the readings of 1st to 4th and 5th to 8th sensors differed by 2.048 milliseconds. The ADC occurrence was pushed to the end, after the 8th sensor so that the consistency among the datasets be restored.
2. The baud rate was increased from 5000 kbps to 1000 kbps. This increased the sampling rate dramatically and gave us more lead over the Nyquist frequency. This also reduced the pulse width for each sensor down to 90 µsec thereby also reducing the minimum separation required between two pulses down to 2000 points, i.e., 108 µsec.
3. Building on point 2, the separation was further optimized with tolerances for more robustness factored in.

Figure 16.9 and the following table enlists the values achieved through optimization.

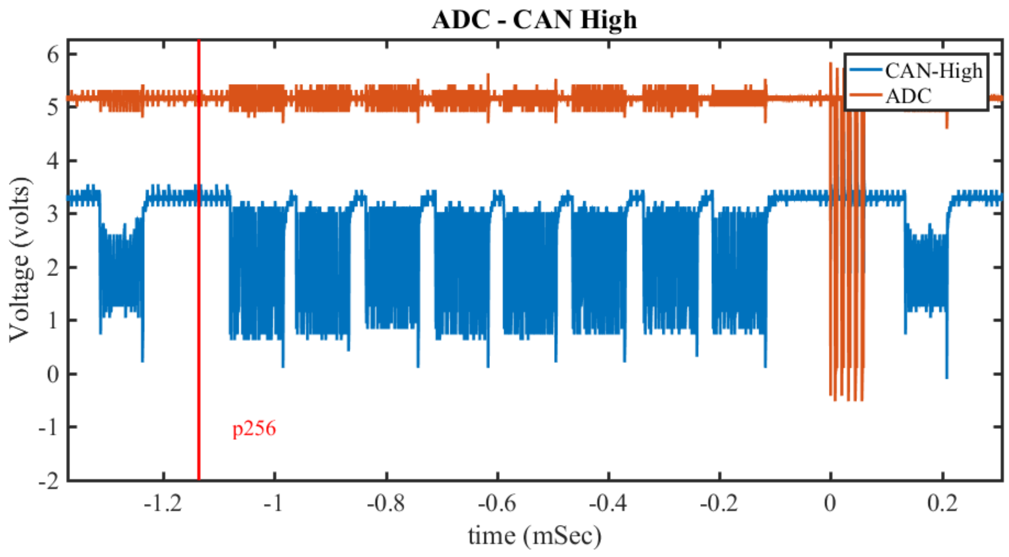


Figure 17.2 : Figure showing the optimized position for ADC and the inter-pulse separation

The attained optimizations are:

Gain in Nyquist Frequency:	More than 120 Hz
Reduction in sampling period:	More than 600 µsec
Reduction in minimum separation required	More than 46% i.e. ≈ 93 µsec (1713 points)
Reduction in pulse width of each sensor	More than 55%, i.e. ≈ 111 µsec

RESULTS AND CONCLUSIONS |

At the end of the project, it was found that there was a trade-off between long-time and short-time robustness. When attempted to make the sensors robust to high magnetic fields, the long-time robustness was lost, and vice versa. As it stands now, it can continue to measure even when subjecting to very high magnetic fields and can continue to do so for more than an hour (several hours in some cases). Given that the required measurement is for few seconds, it should be more than adequate for the project but of course, there remains scope for future improvements. As it stands now, the sensors demonstrate the following mentioned properties:

Duration of set/reset pulse	0.796 ± 0.168 μsec.
Mean Sensitivity	35.97 μG/digital level
Average Noise Density	0.84 Signal/$\sqrt{\text{Hz}}$
Average RMS Noise	0.97 Signal
Maximum RMS Noise	36.15 μG
Maximum Spectral White Noise Density	3.11 μG/√Hz
Prescaler of Position Counter	1
Prescaler of ADC Counter	256
Prescaler of Sync Timer	256
Scheme	Bipolar
Baud Rate	1 Mbps
Bit rate	500 kbps
Minimum Separation	108 μsec, 2000 points
Pulse Width	90 μsec
Sampling Counter	180
ADC Counter	70
Sampling Time	1448 μsec
Sampling Frequency	≈ 691 Hz
Nyquist Frequency	≈ 345 Hz
Gain in Nyquist Frequency:	More than 120 Hz
Reduction in sampling period:	More than 600 μsec
Reduction in minimum separation required	More than 46% i.e. ≈ 93 μsec (1713 points)
Reduction in pulse width of each sensor	More than 55%, i.e. ≈ 111 μsec

APPENDICES |

APPENDIX: A

OBJECTIVE 1 | THE INTERFACING PROGRAM |

A.1

Directories:

- candll
  - Executable directories: $(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);$(MSBuild_ExecutablePath);$(SystemRoot)\SysWow64;$(FxCopDir);$(PATH);
  - Include directories: $(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(WindowsSDK_IncludePath);$(WindowsSdkDir)include;$(FrameworkSDKDir)\include;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Include;C:\Program Files (x86)\Windows Kits\10\Include\10.0.15063.0\ucrt;
  - Reference directories:

    $(VC_ReferencesPath_x86);
  - Library directories: $(VCInstallDir)lib;$(VCInstallDir)atlmfc\lib;$(WindowsSdkDir)lib;$(FrameworkSDKDir)\lib;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Lib;C:\Program Files (x86)\Windows Kits\10\Lib\10.0.15063.0\ucrt\x86;
  - Library WinRT Directories:

    $(WindowsSDK_MetadataPath);
  - Source Directories:

    $(VC_SourcePath);
  - Exclude Directories: $(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC_LibraryPath_x86);
- GSPC
  - Executable directories: $(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);$(MSBuild_ExecutablePath);$(SystemRoot)\SysWow64;$(FxCopDir);$(PATH);
  - Include directories: $(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(WindowsSdkDir)include;$(WindowsSDK_IncludePath);$(FrameworkSDKDir)\include;.\;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\include;C:\Program Files (x86)\Windows Kits\10\Include\10.0.15063.0\ucrt;

- Reference directories:

  $(VC_ReferencesPath_x86);

- Library directories:
  $(VCInstallDir)lib;$(VCInstallDir)atlmfc\lib;$(WindowsSdkDir)lib;$(FrameworkSDKDir)\lib;Debug\;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Lib;C:\Program Files (x86)\Windows Kits\10\Lib\10.0.15063.0\ucrt\x86;

- Library WinRT Directories:

  $(WindowsSDK_MetadataPath);

- Source Directories:

  $(VC_SourcePath);

- Exclude Directories:
  $(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC_LibraryPath_x86);

APPENDIX: B

OBJECTIVE 2 | THE ENCODING – DECODING SCHEME |

B.1

```
//(SWPCDIg, line 116 to 117)
short data[8][3];
short KCprevdata[8][3];

//(SWPCDIg, line 3147 to 3166)
void CSWPCDlg::Send_Short( unsigned char command, short var)
{
  unsigned char data[3];
  data[0]= command;          //cmd
  data[1]= (unsigned char)var;
  data[2]= (unsigned char)(var>>8);

  //DLL
  CAN_SendData(pCAN, CAN_device_ID, 0, data, 3);
}

void CSWPCDlg::Send_All_Short(unsigned char command,  short var)
{
   unsigned char data[3], negativedata[3];
  data[0]= command;          //cmd
  data[1]= ( unsigned char)var;
  data[2]= ( unsigned char)(var>>8);
  unsigned short ii,jj;
   short negvar;


//(SWPCDIg, line 3828)

void CSWPCDlg::SignalProc( short* data, double* outp)// Chang Chi
(20130109)
```

```
//(SWPCDIg, line 4178 to 4185)

void CSWPCDlg::OnBnClickedFileoutput()
{
	// TODO: Add your control notification handler code here

	wseriessize = seriessamp * nbseries * nbloop;
	writebuffer = (short*) calloc(24 * wbuffsize,sizeof(short));
```

B.2

```
//ADC AD 7682

(2<<INCC_H)| // 2 = Bipolar ; Inx referenced to COM = VREF/2 ± 0.1 V
```

APPENDIX: C

OBJECTIVE 3 | THE SENSOR FIRMWARE |

C.1

IAR Embedded Workbench

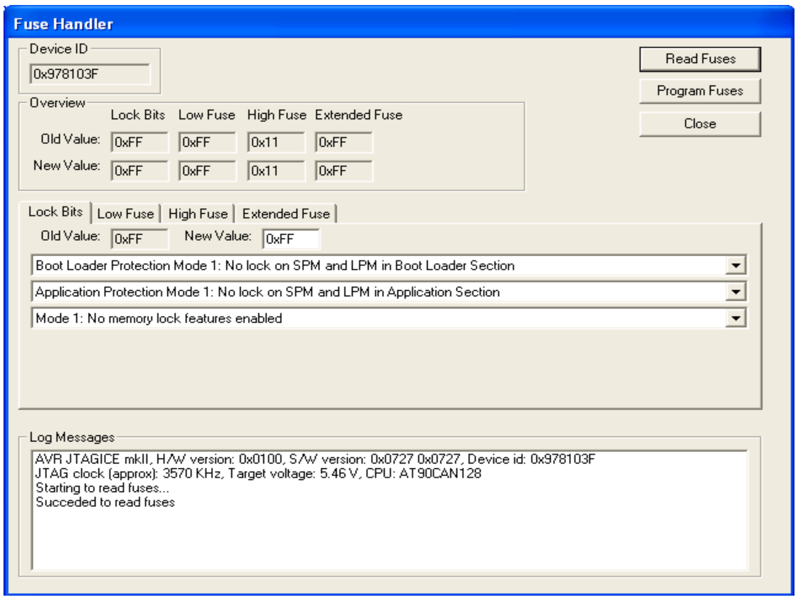
- Version: 5.3 (installed in Windows XP mode)
- Prerequisites: AVR studios 5.1, Windows XP mode,
- Flashed with JTAGICE MKII
- Tested on so7, s14, new sensor (magnetic sensor 2)
- Locations of sensor AVR programs (for admin-PC beside the printer) in XP mode
  - Old sensor: C:\sensor v2 5.30
  - New sensor: C:\sensor V3

Steps to make the IAR Embedded Workbench work

1. Clean the registry and cache using CCleaner or equivalent
2. Install in Windows XP mode
3. After installation, from start -> IAR Systems -> IAR Systems license activation
4. In there, next-> do not register any product right now

C.2

For new sesors:



Device ID: 0x978103F

Overview

	Lock Bits	Low Fuse	High Fuse	Extended Fuse
Old Value:	0xFF	0xFF	0x11	0xFF
New Value:	0xFF	0xFF	0x11	0xFF

Lock Bits | Low Fuse | High Fuse | Extended Fuse

Old Value: 0xFF New Value: 0xFF

Boot Loader Protection Mode 1: No lock on SPM and LPM in Boot Loader Section

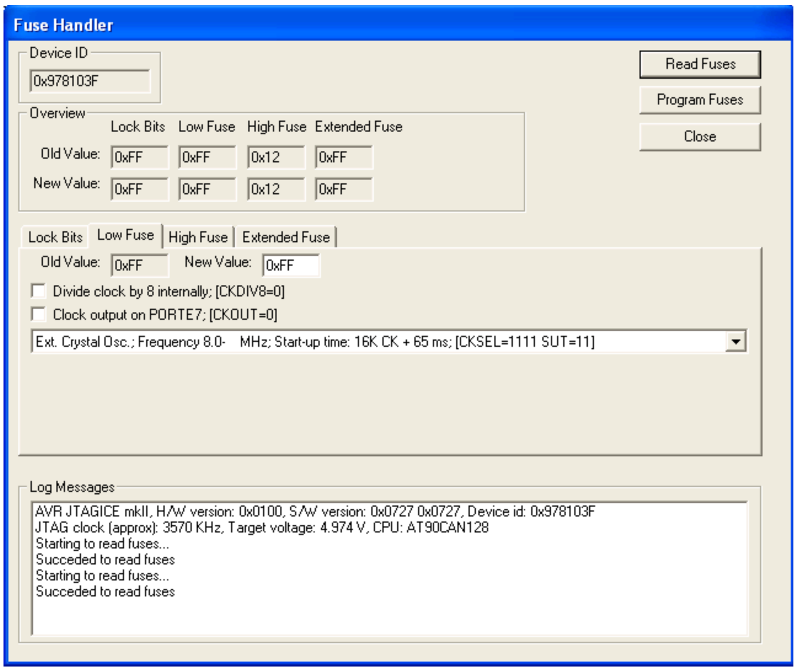
Application Protection Mode 1: No lock on SPM and LPM in Application Section

Mode 1: No memory lock features enabled

Log Messages

```
AVR JTAGICE mkII, H/W version: 0x0100, S/W version: 0x0727 0x0727, Device id: 0x978103F
JTAG clock (approx): 3570 KHz, Target voltage: 5.46 V, CPU: AT90CAN128
Starting to read fuses...
Succeded to read fuses
```

For old sensors:



Device ID: 0x978103F

Overview

	Lock Bits	Low Fuse	High Fuse	Extended Fuse
Old Value:	0xFF	0xFF	0x12	0xFF
New Value:	0xFF	0xFF	0x12	0xFF

Lock Bits | Low Fuse | High Fuse | Extended Fuse

Old Value: 0xFF New Value: 0xFF

☐ Divide clock by 8 internally; [CKDIV8=0]

☐ Clock output on PORTE7; [CKOUT=0]

Ext. Crystal Osc.; Frequency 8.0- MHz; Start-up time: 16K CK + 65 ms; [CKSEL=1111 SUT=11]

Log Messages

```
AVR JTAGICE mkII, H/W version: 0x0100, S/W version: 0x0727 0x0727, Device id: 0x978103F
JTAG clock (approx): 3570 KHz, Target voltage: 4.974 V, CPU: AT90CAN128
Starting to read fuses...
Succeded to read fuses
Starting to read fuses...
Succeded to read fuses
```

APPENDIX: D

THE EXPERIMENTAL PHASE

D.1

```
clc
close all
clear all


trs1 = xlsread('New sensor reset.xlsx', 'D1:D2500');
vrs1 = xlsread('New sensor reset.xlsx', 'E1:E2500');
ts1 = xlsread('New sensor set.xlsx', 'D1:D2500');
vs1 = xlsread('New sensor set.xlsx', 'E1:E2500');

figure
plot(trs1,vrs1,'r.')
hold on;
plot (ts1,vs1,'b.')
xlabel('time')
ylabel('Voltage')
title ('New Sensor')

trs2 = xlsread('Old sensor reset.xlsx', 'D1:D2500');
vrs2 = xlsread('Old sensor reset.xlsx', 'E1:E2500');
ts2 = xlsread('Old_sensor set.xlsx', 'D1:D2500');
vs2 = xlsread('Old_sensor set.xlsx', 'E1:E2500');

figure
plot(trs2,vrs2,'r.')
hold on;
plot (ts2,vs2,'b.')
xlabel('time')
ylabel('Voltage')
title('Old Sensor')

figure
diff1=(vrs1+vs1);
mean1=mean(vrs1+vs1);
plot(trs1,diff1,'b.')
hold on
plot(trs1,mean1,'r.')
hold on
plot(ts1,diff1,'b.')
hold on
plot(ts1,mean1,'y.')
xlabel('Time')
ylabel('Voltage')
title('New Sensor Difference in VPP')

figure
diff2=(vrs2+vs2);
mean2=mean(vrs2+vs2);
plot(trs2,diff2,'b.')
hold on
```

```
plot(trs1,mean1,'r.')
hold on
plot(ts2,diff2,'b.')
hold on
plot(ts2,mean2,'y.')
xlabel('Time')
ylabel('Voltage')
title('Old Sensor Difference in VPP')
```

APPENDIX: E

OBJECTIVE 4 | TO DETERMINE THE SENSITIVITY |

E.1

PLOT DATA:

```
clc
close all;
clear all;

d = load('data00009.txt');// could be any file name
Column_Start=1;
    for jj=1:32
        temp=d(1,jj);
        if (temp~=0)
            Column_Start=jj;
            break
        end
    end
x=d(:,Column_Start);
y=d(:,Column_Start+1);
z=d(:,Column_Start+2);
c=d(:,Column_Start+3);
offsetx = (x(2:end) + x(1:end-1)) /2;
offsety = (y(2:end) + y(1:end-1)) /2;
offsetz = (z(2:end) + z(1:end-1)) /2;

figure
plot(x,'b')
hold on
plot(offsetx,'r')
legend('Field','Bridge Offset')
title('X')
figure
plot(y,'b')
hold on
plot(offsety,'r')
legend('Field','Bridge Offset')
title('Y')
figure
plot(z,'b')
hold on
plot(offsetz,'r')
legend('Field','Bridge Offset')
title('Z'
```

E.2

The X and Y axis both are digital levels

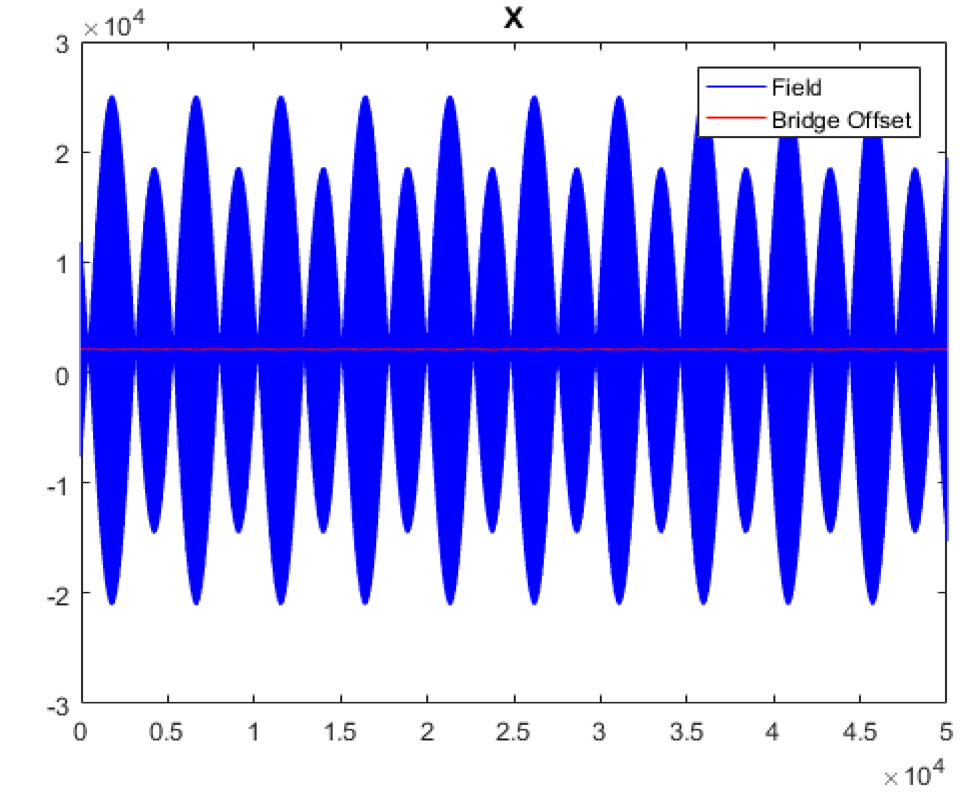


Figure 4.1: S17, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

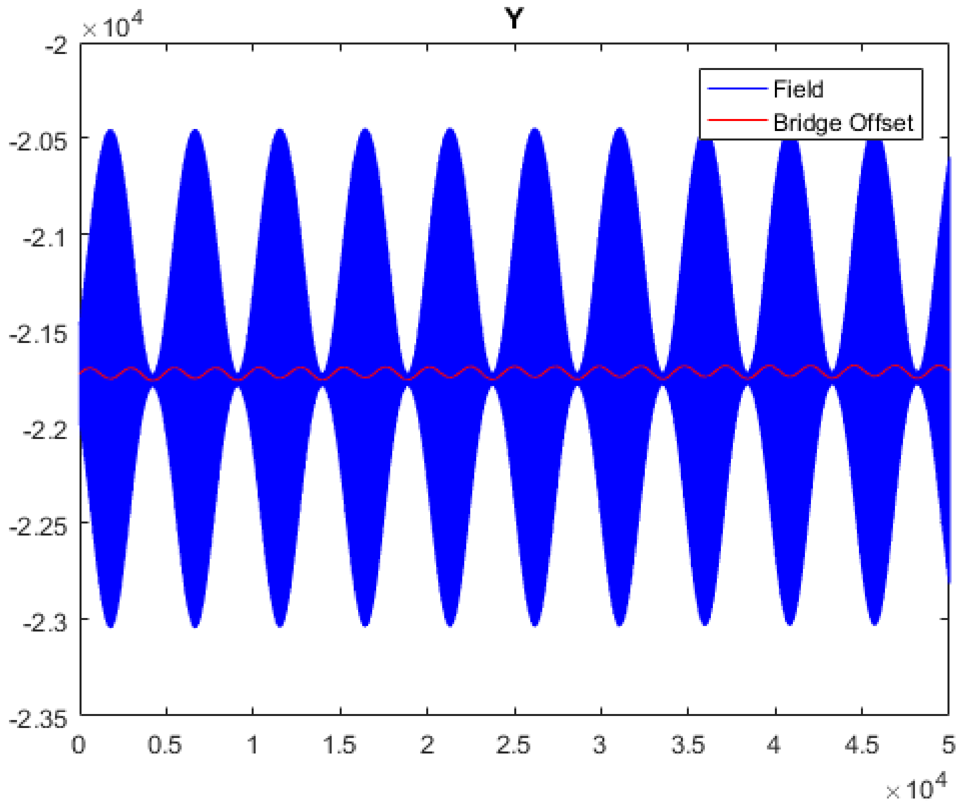


Figure 4.2: S17, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

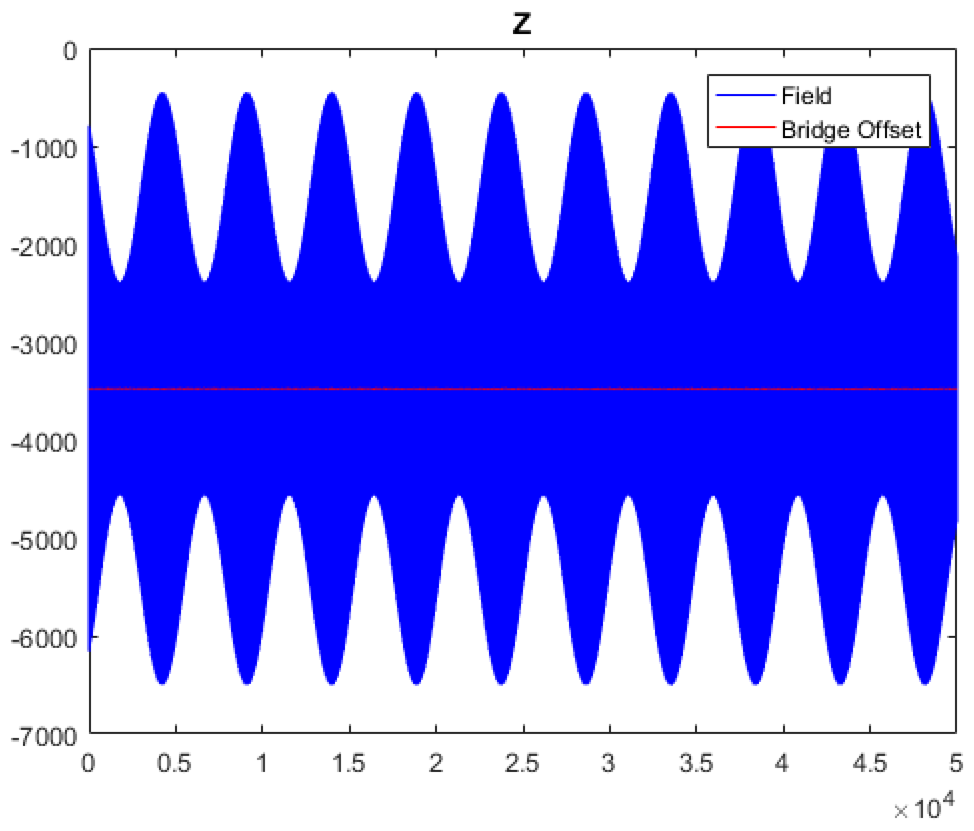


Figure 4.3: S17, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

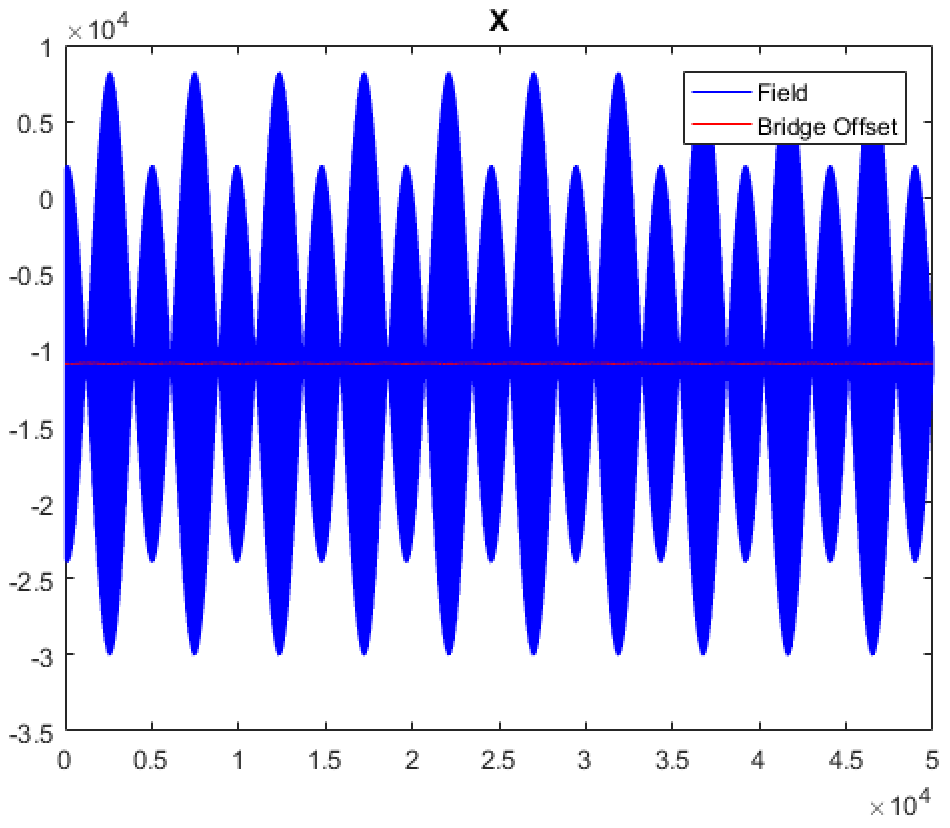


Figure 4.4: S7, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

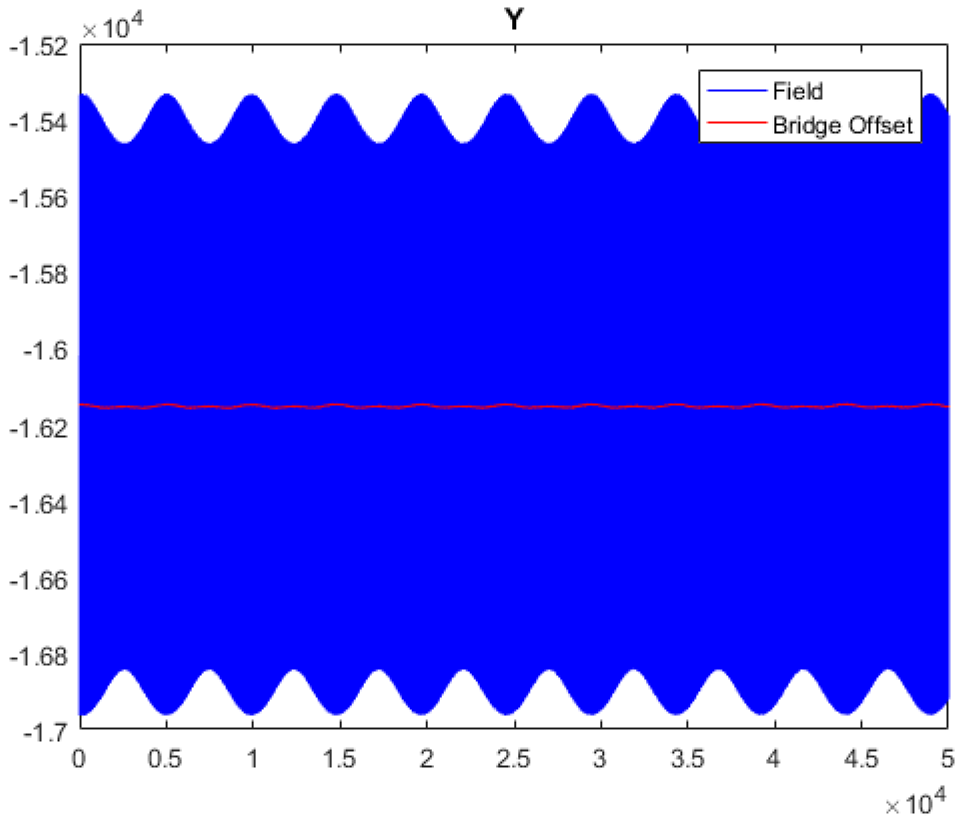


Figure 4.5: S7, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

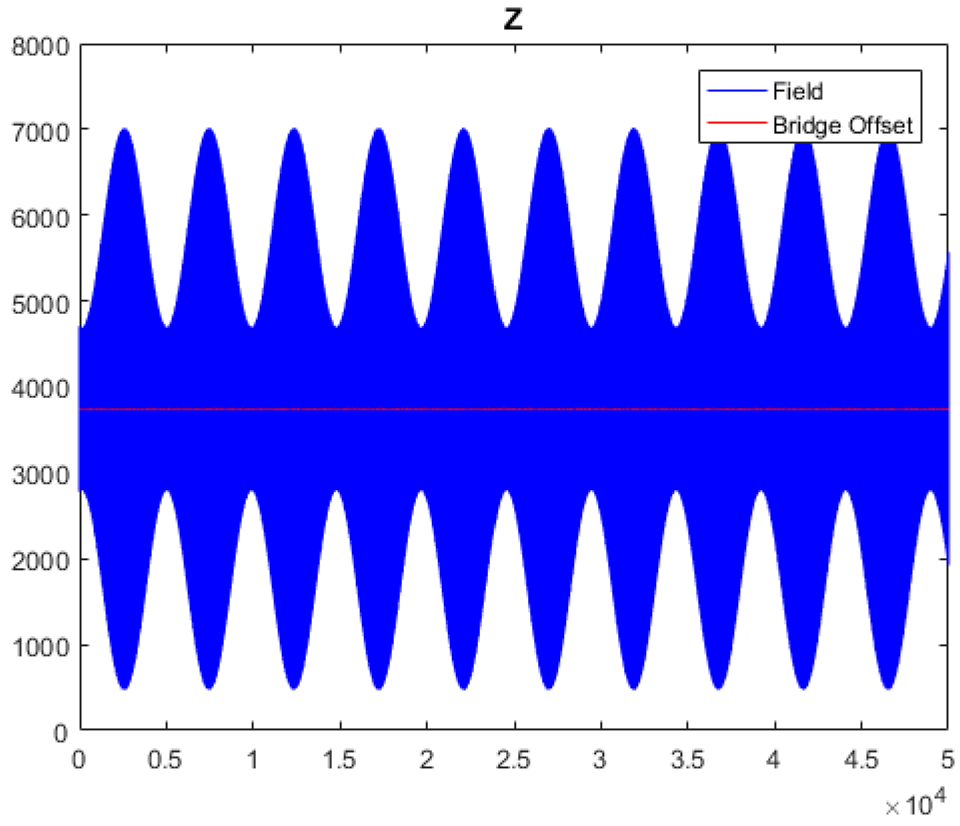


Figure 4.6: S7, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

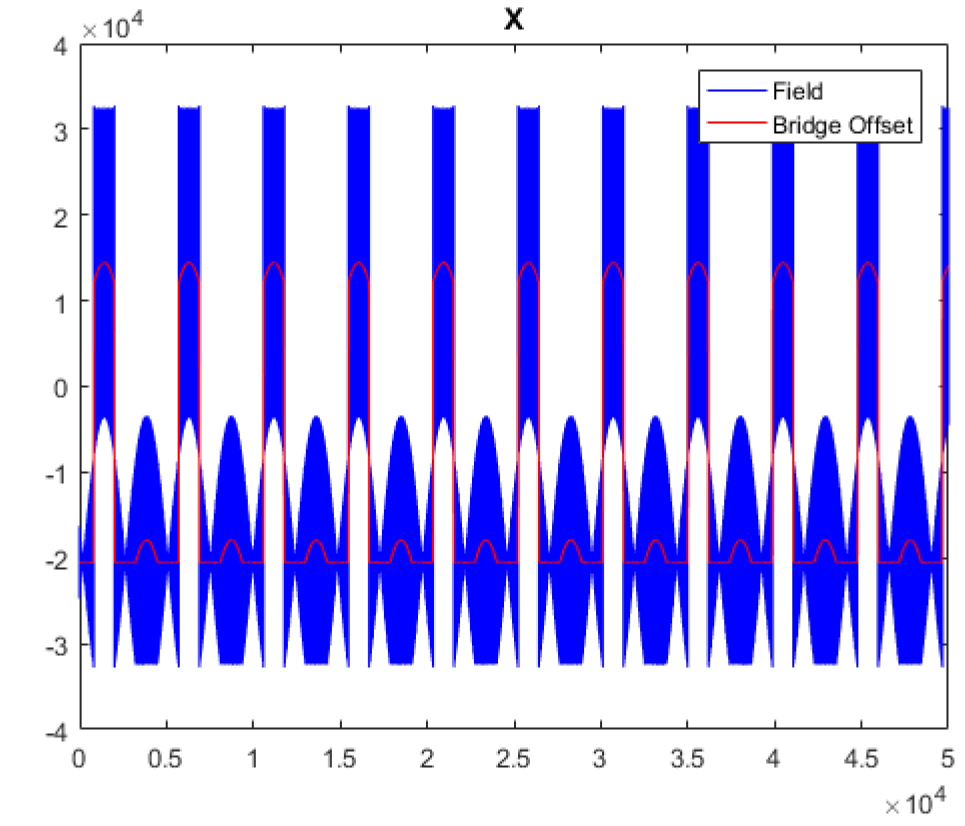


Figure 4.7: S5, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

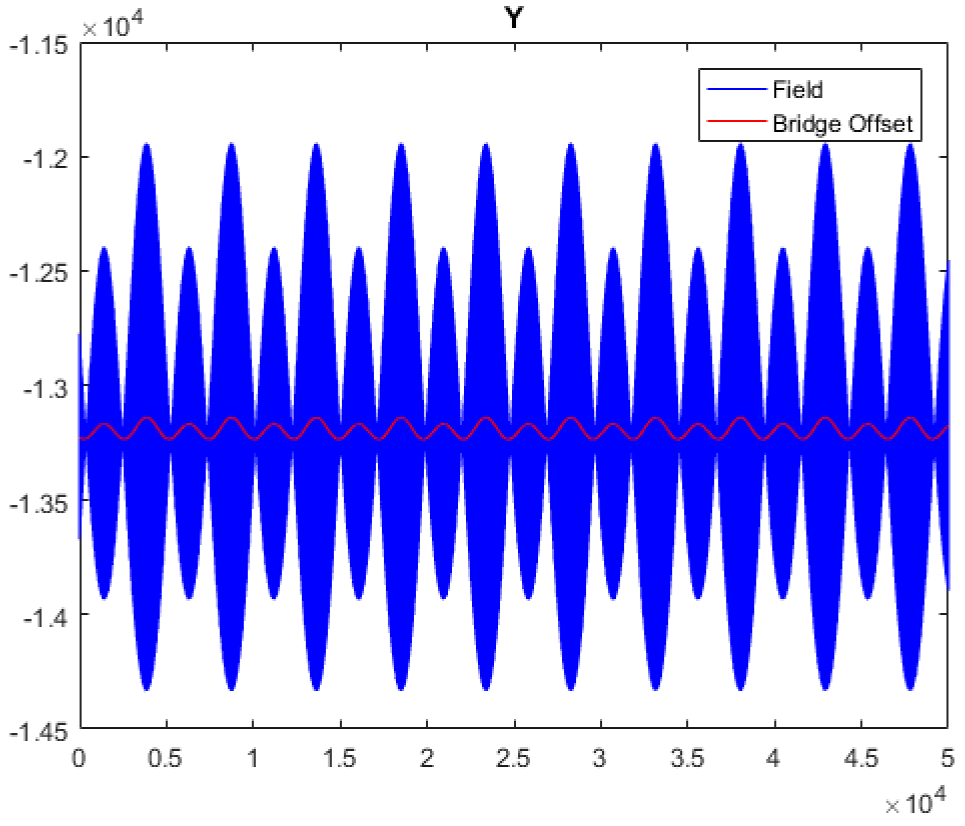


Figure 4.8: S5, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

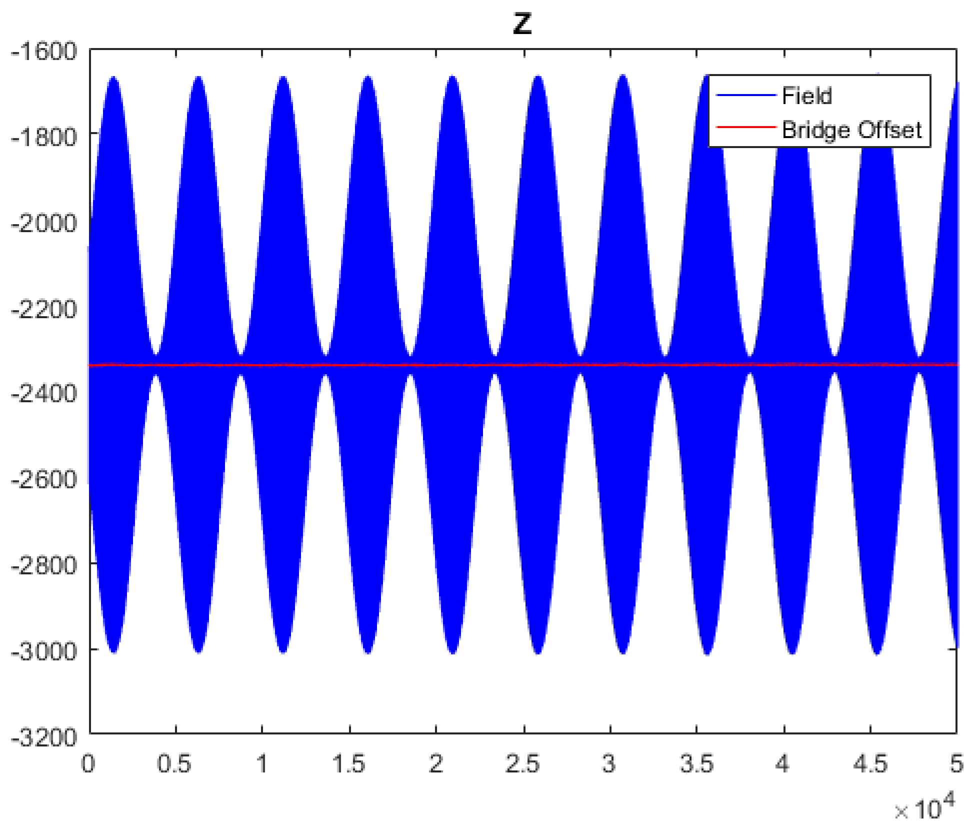


Figure 4.9: S5, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

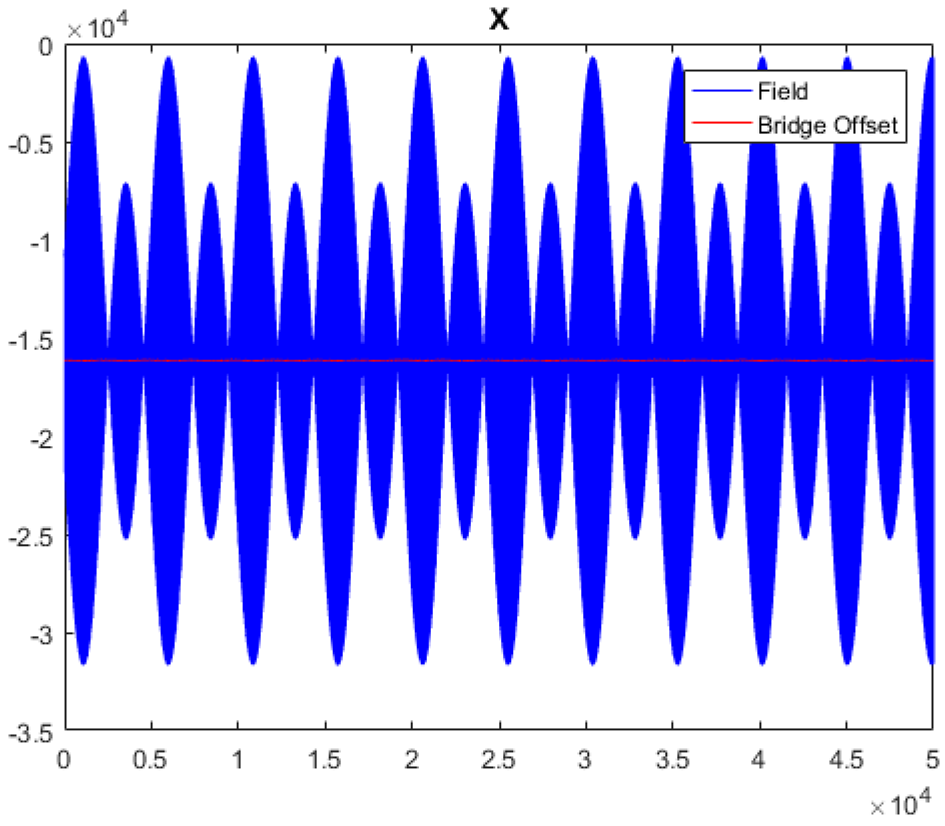


Figure 5.0 : S14, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

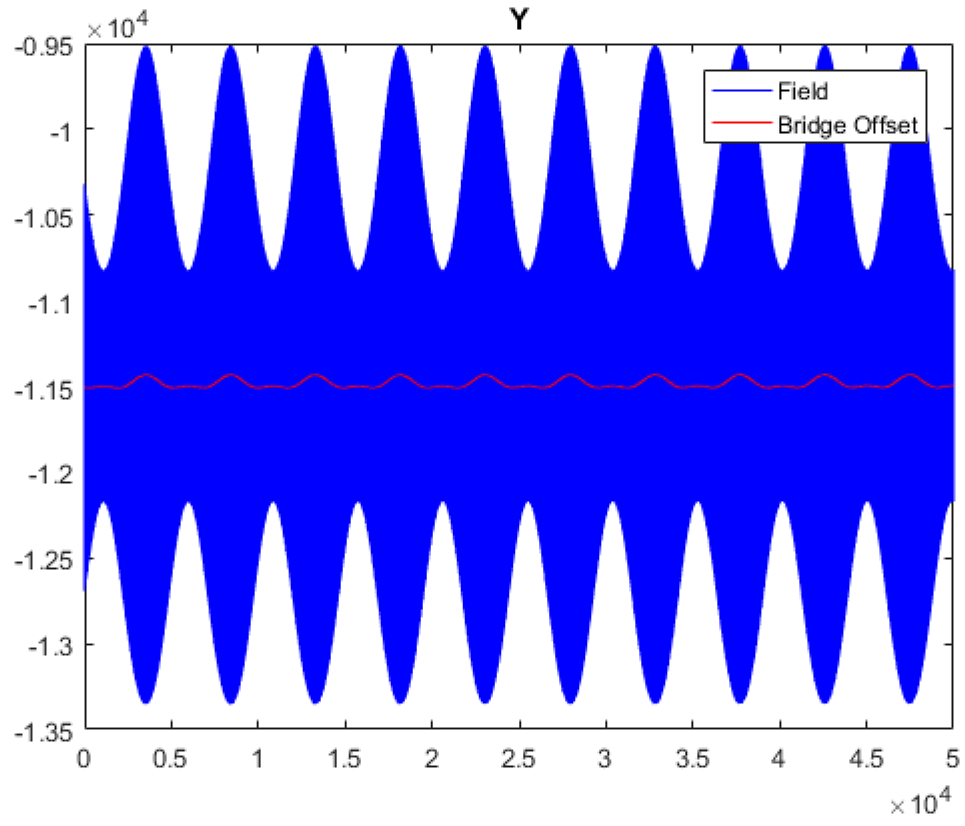


Figure 5.1: S14, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

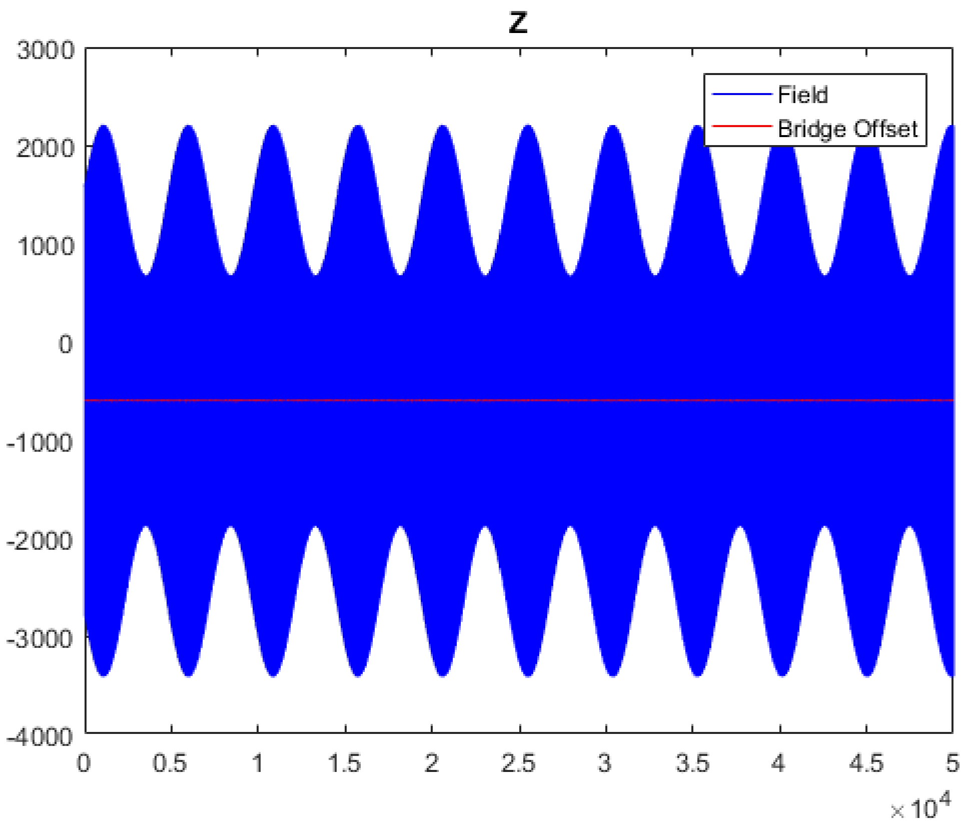


Figure 5.2: S14, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

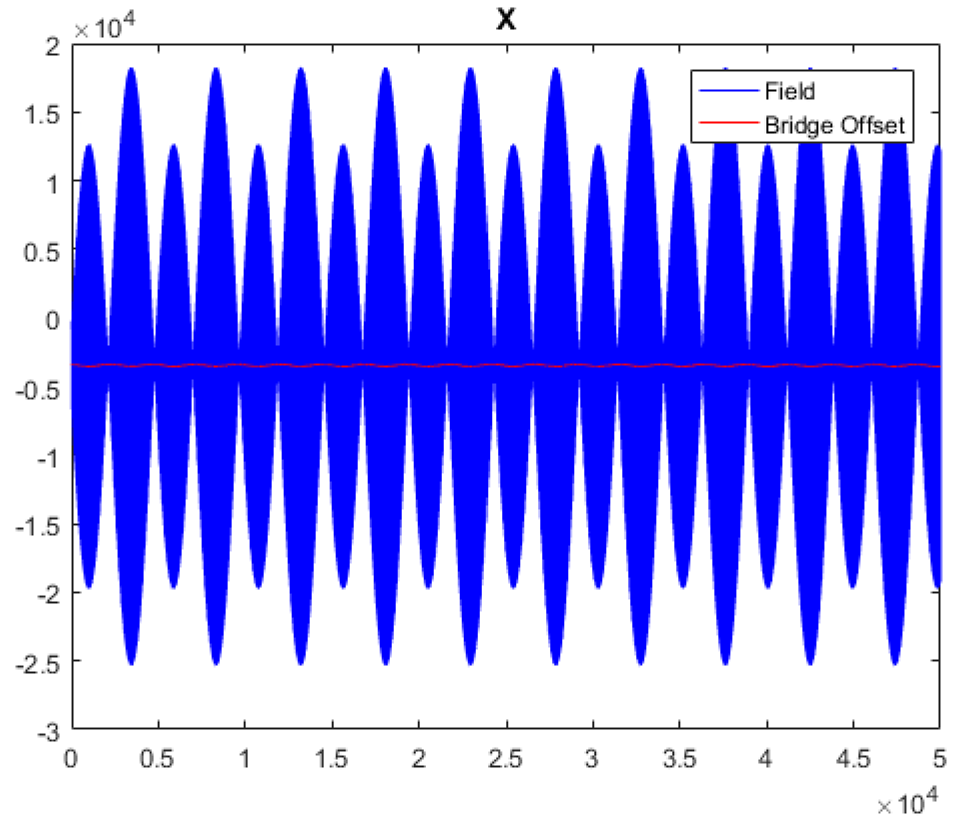


Figure 5.3: S10, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

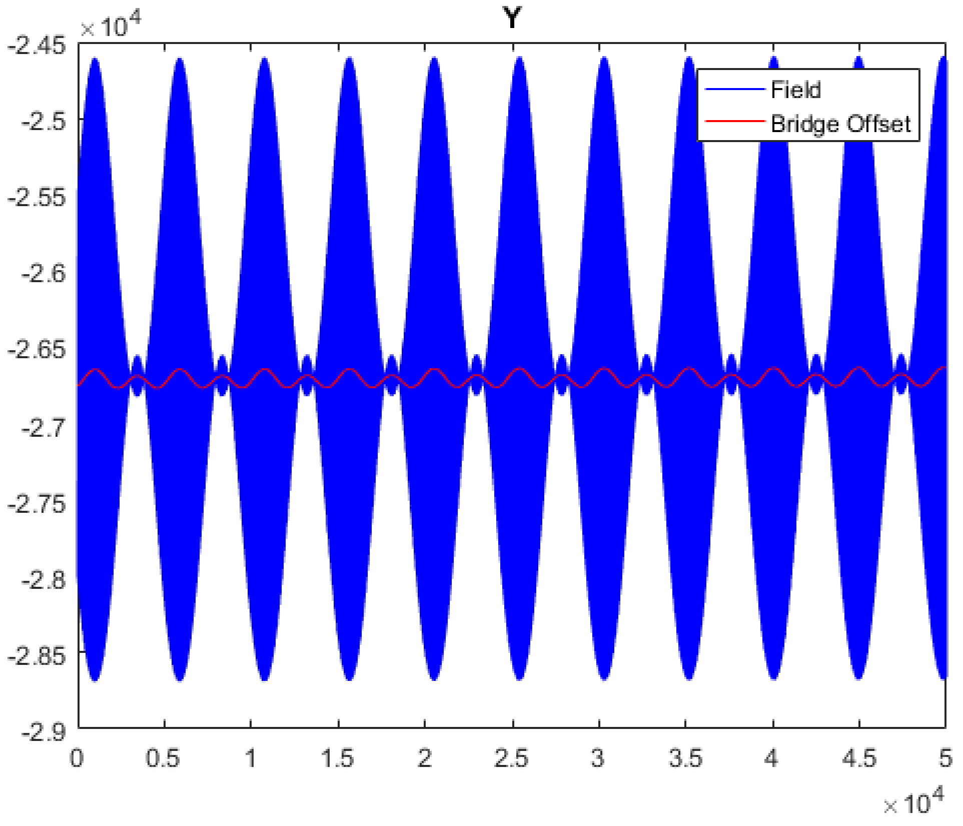


Figure 5.4: S10, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

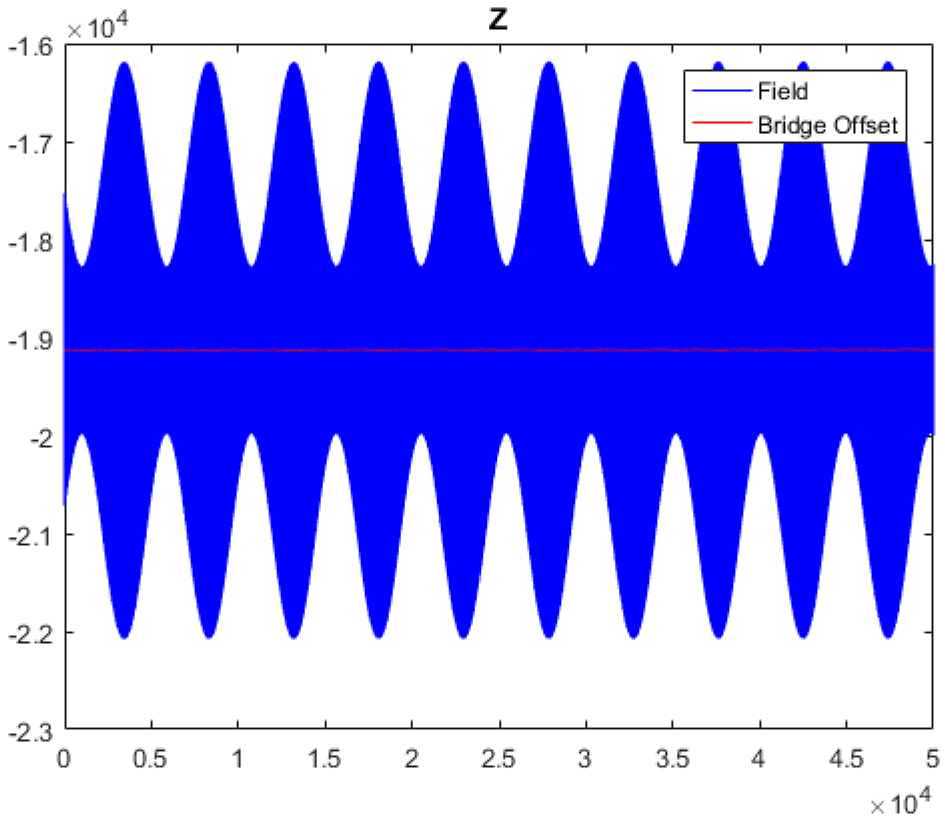


Figure 5.5: S10, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

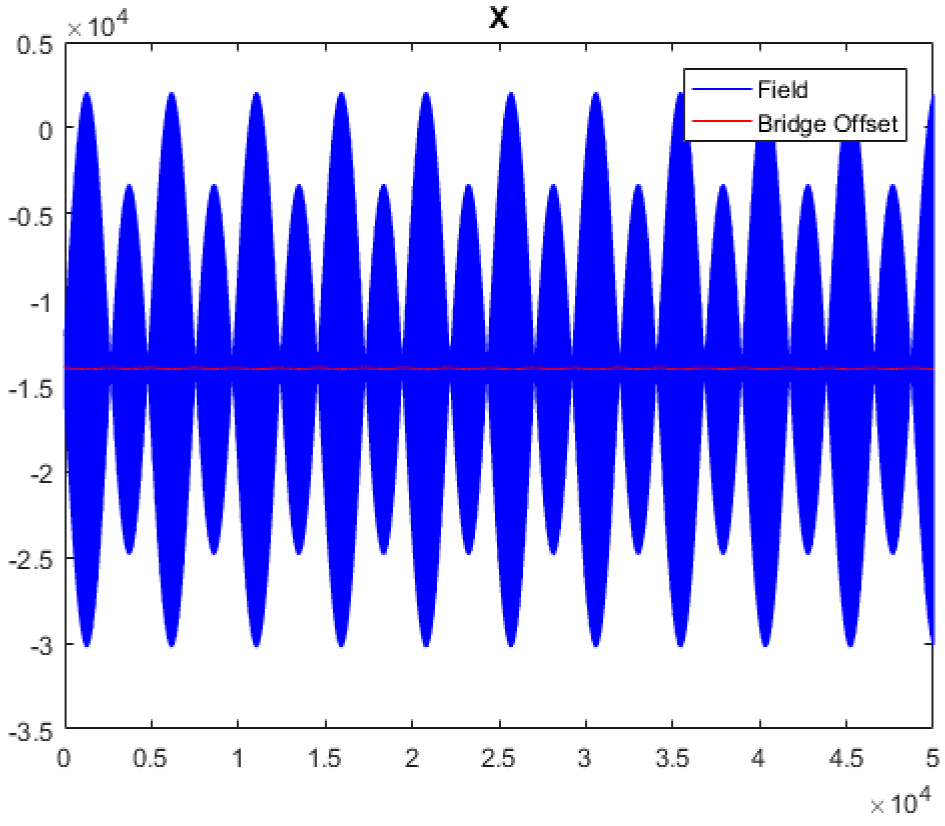


Figure 5.6: S6, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

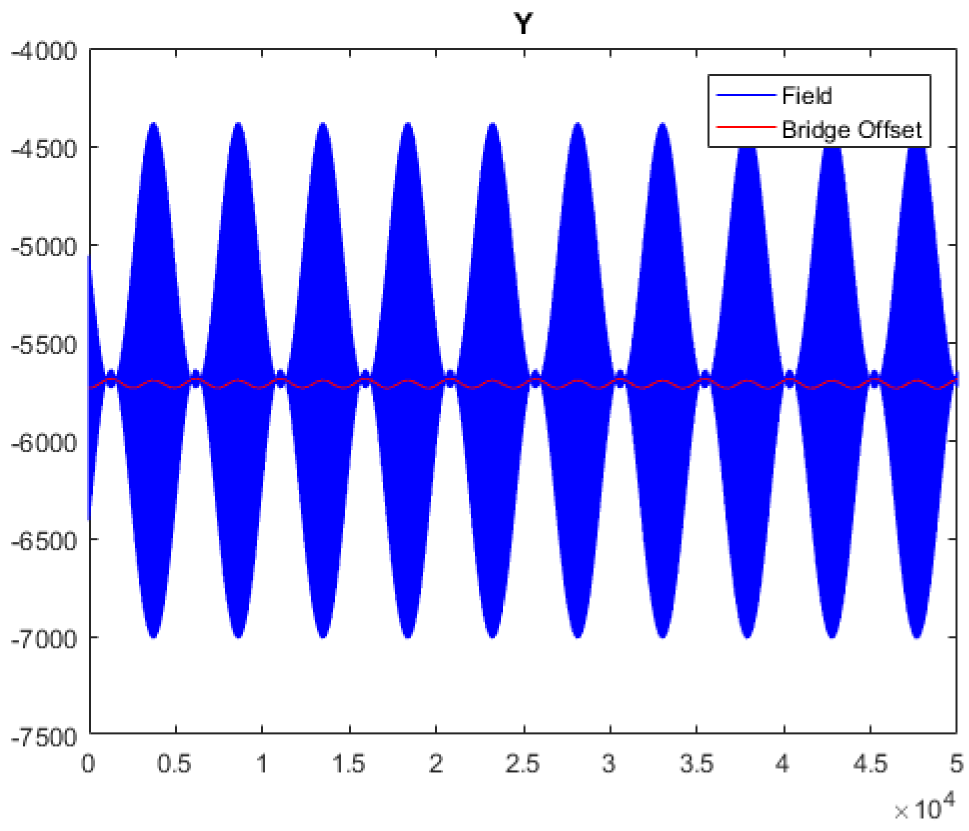


Figure 5.7: S6, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

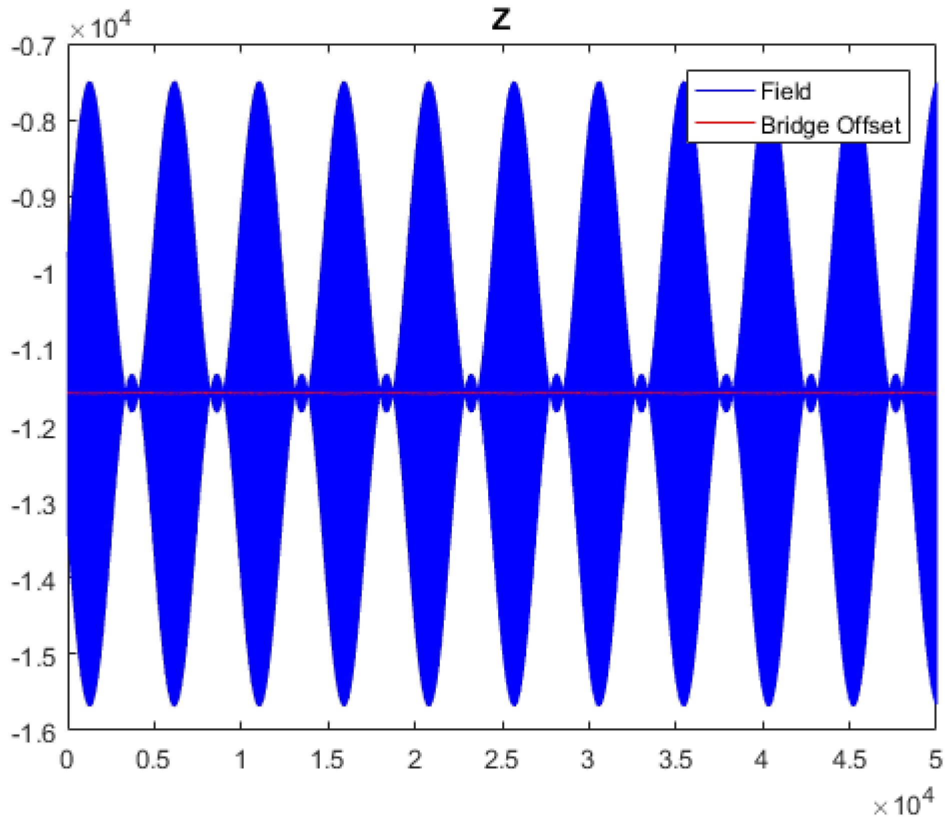


Figure 5.8: S6, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

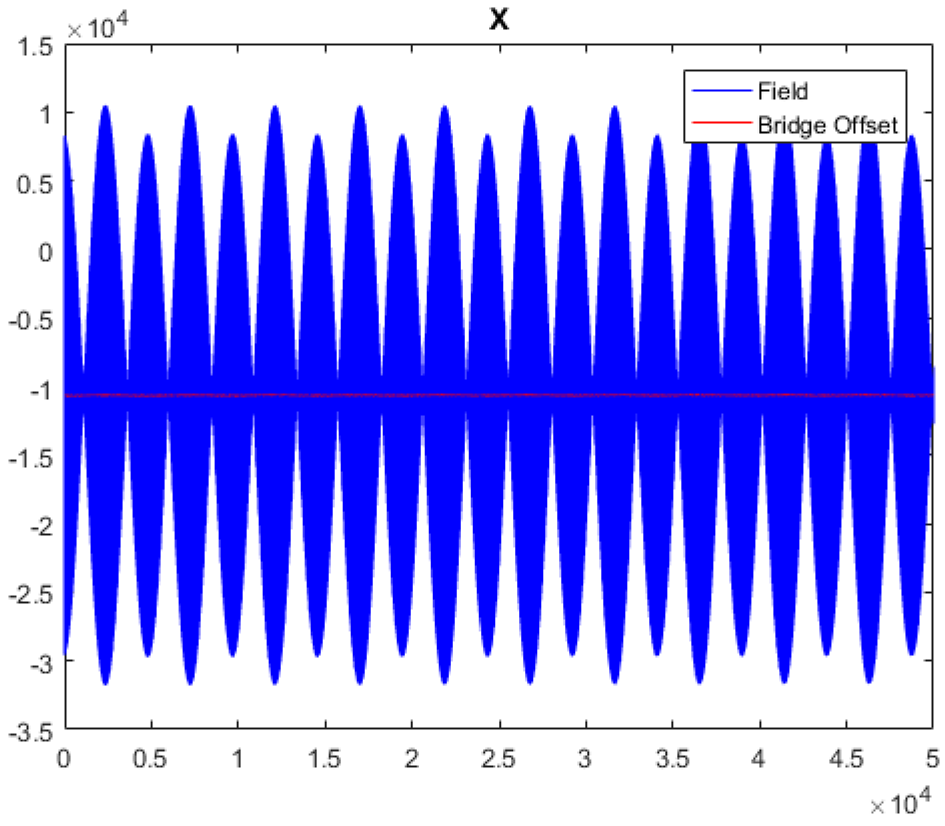


Figure 5.9: S8, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

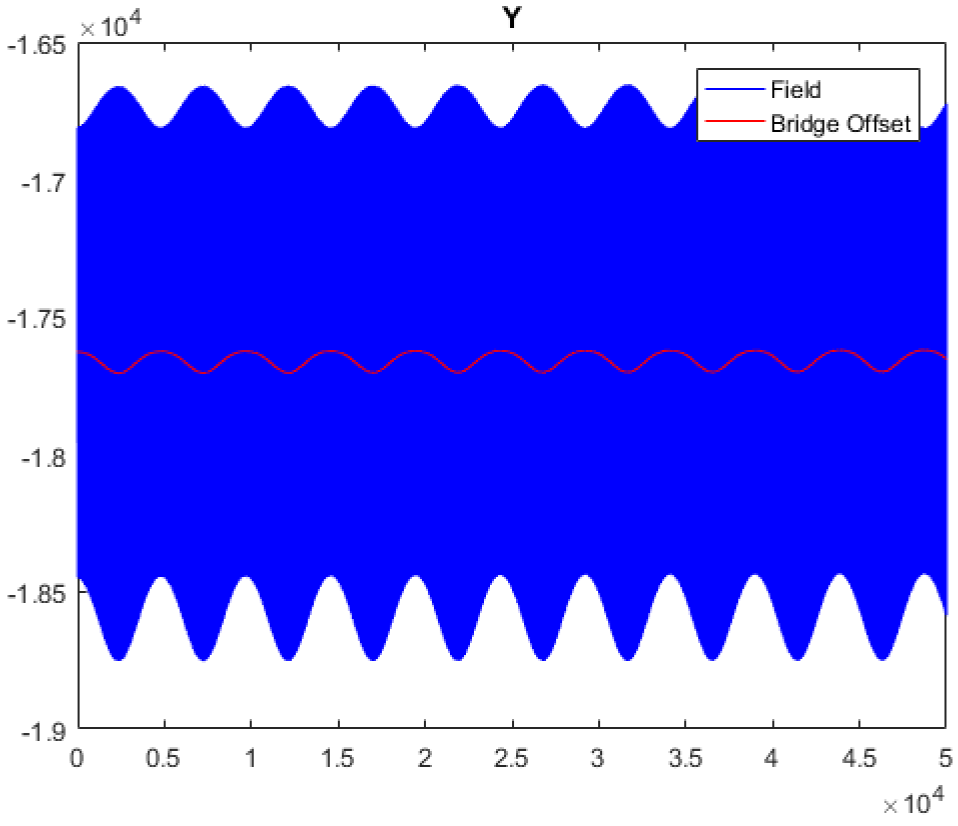


Figure 6.0: S8, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

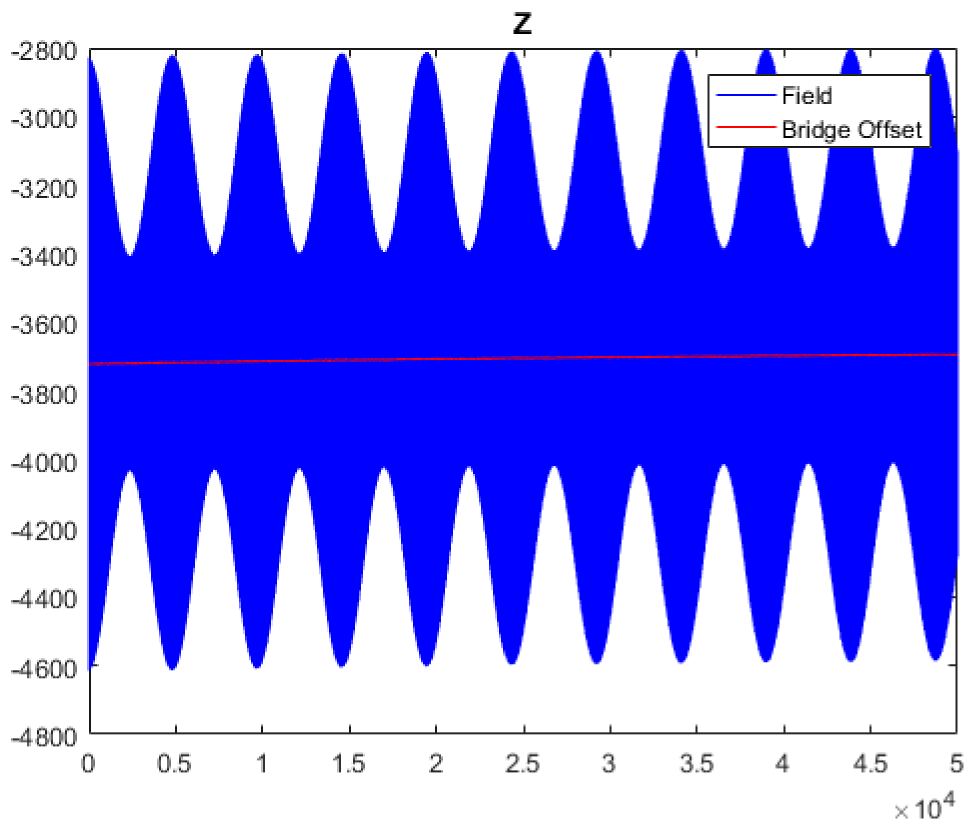


Figure 6.1: S8, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

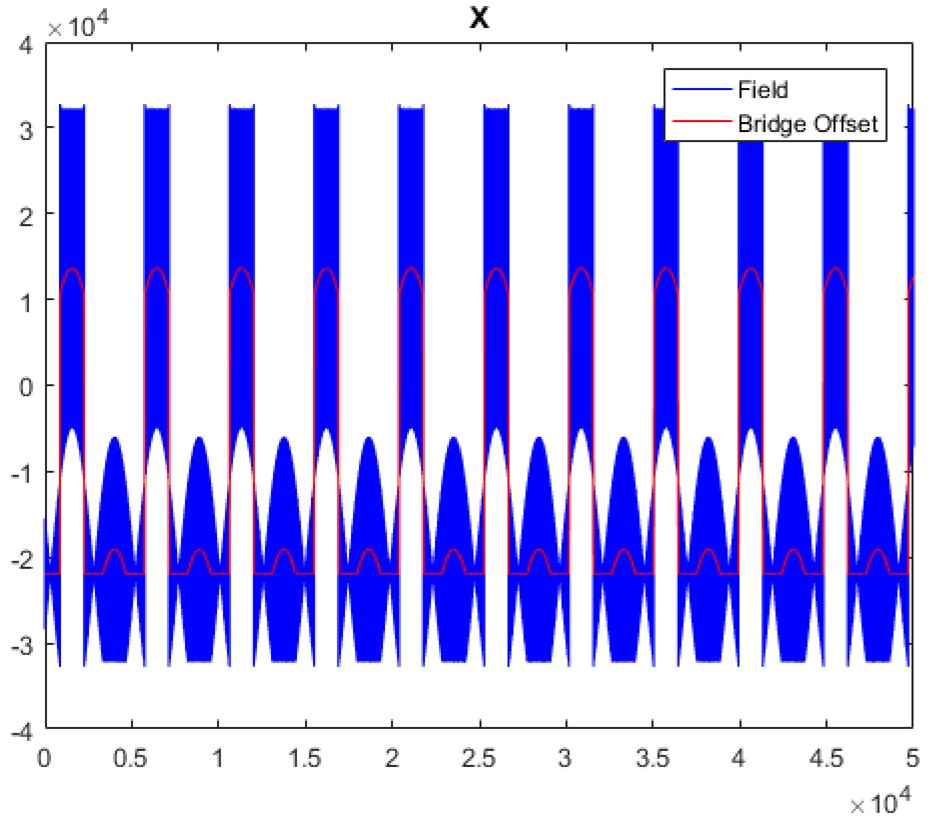


Figure 6.2: S1, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

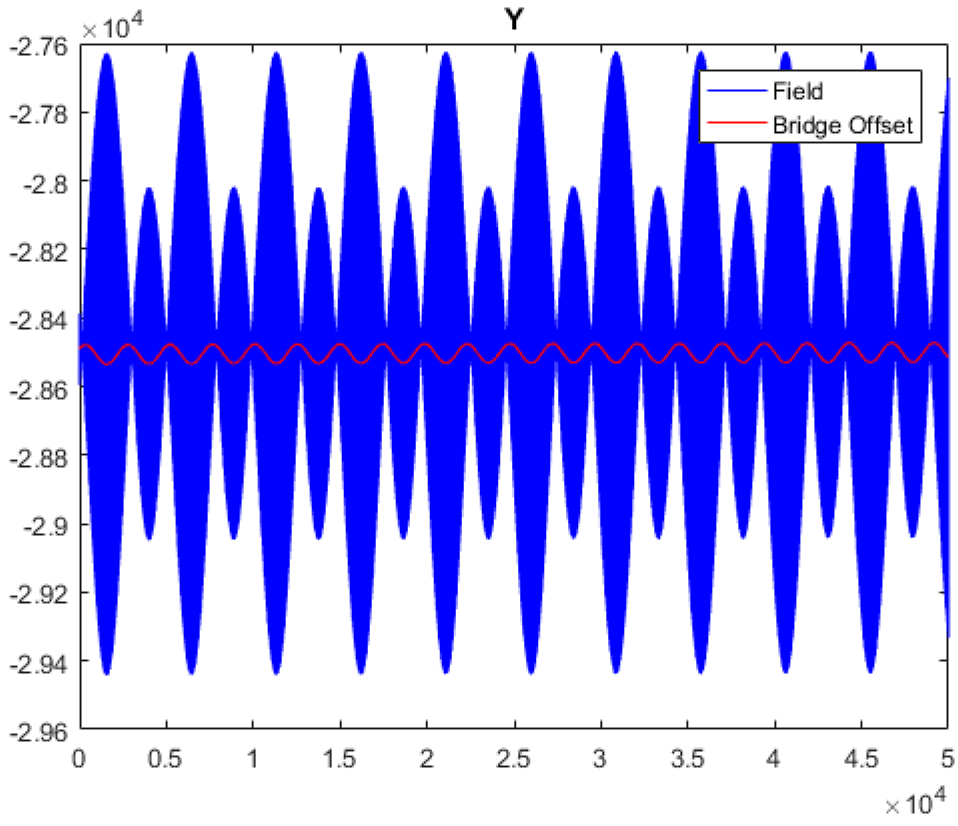


Figure 6.3: S1, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

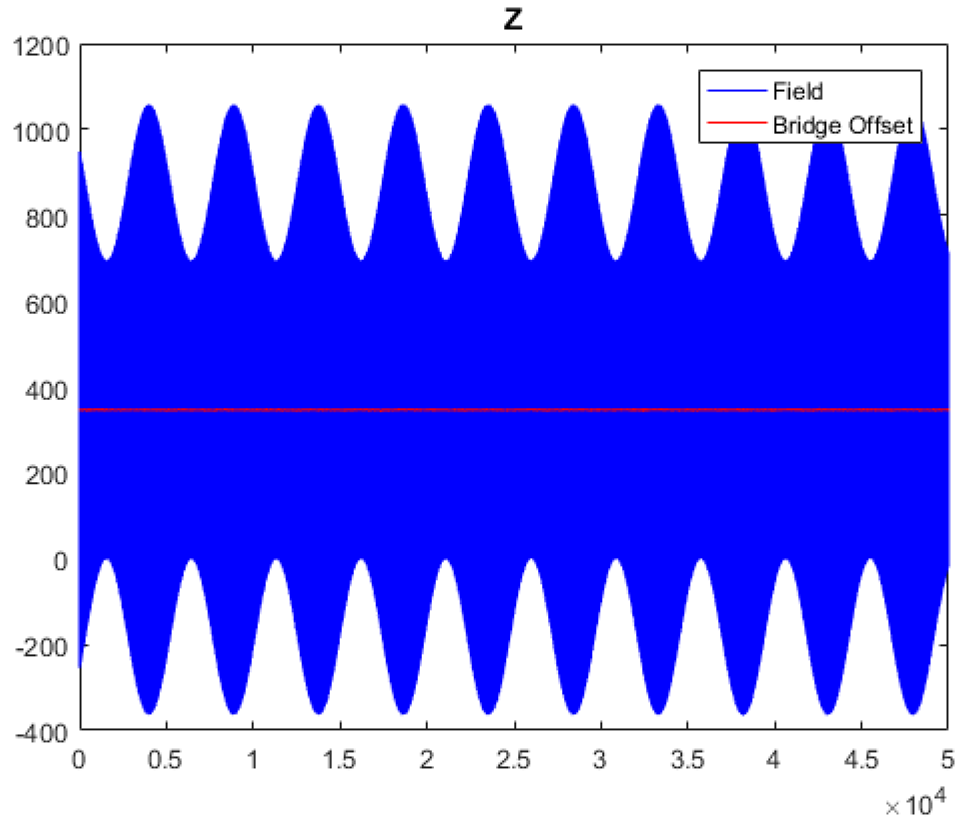


Figure 6.4: S1, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

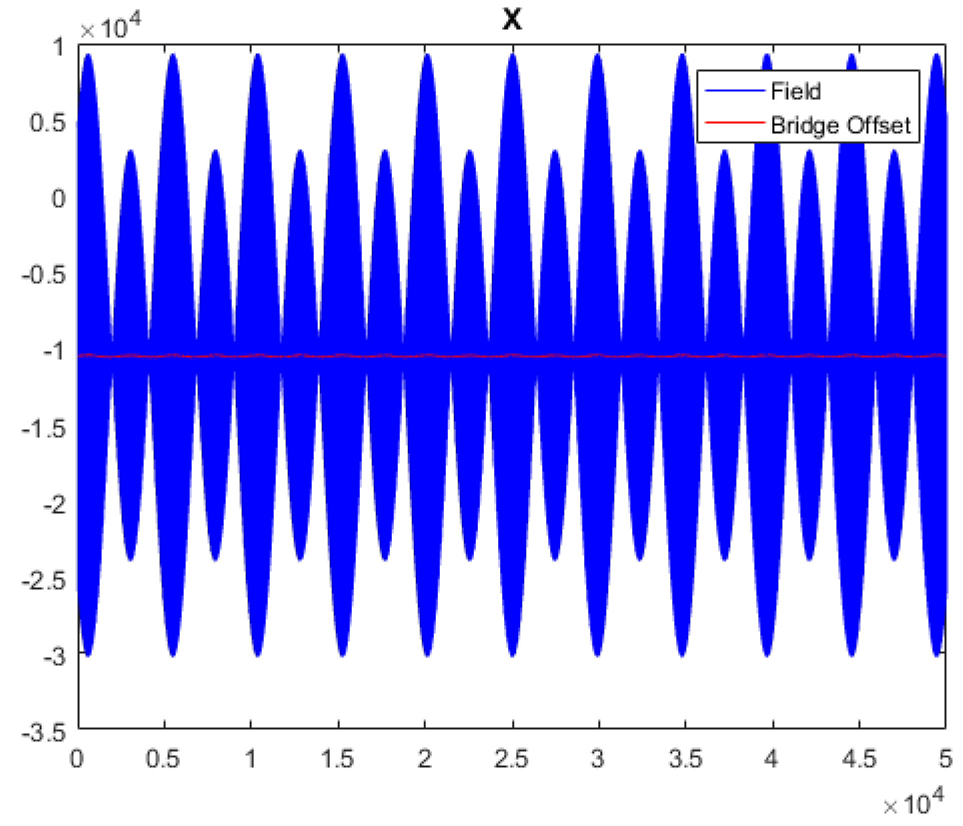


Figure 6.5: S16, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

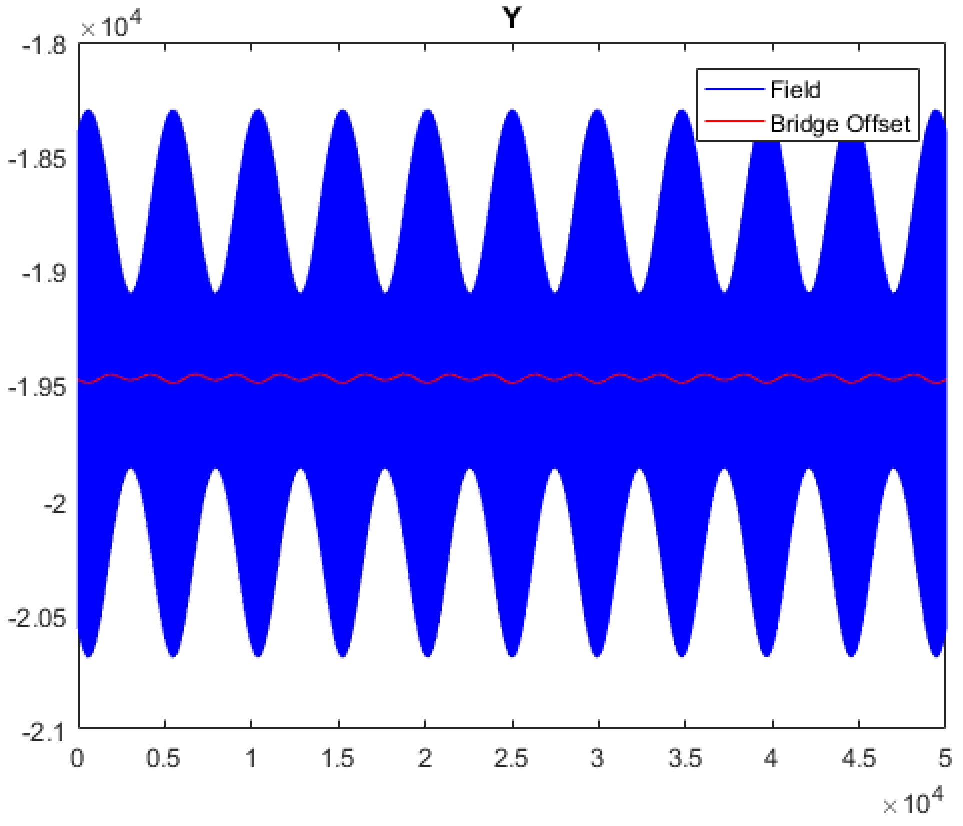


Figure 6.6: S16, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

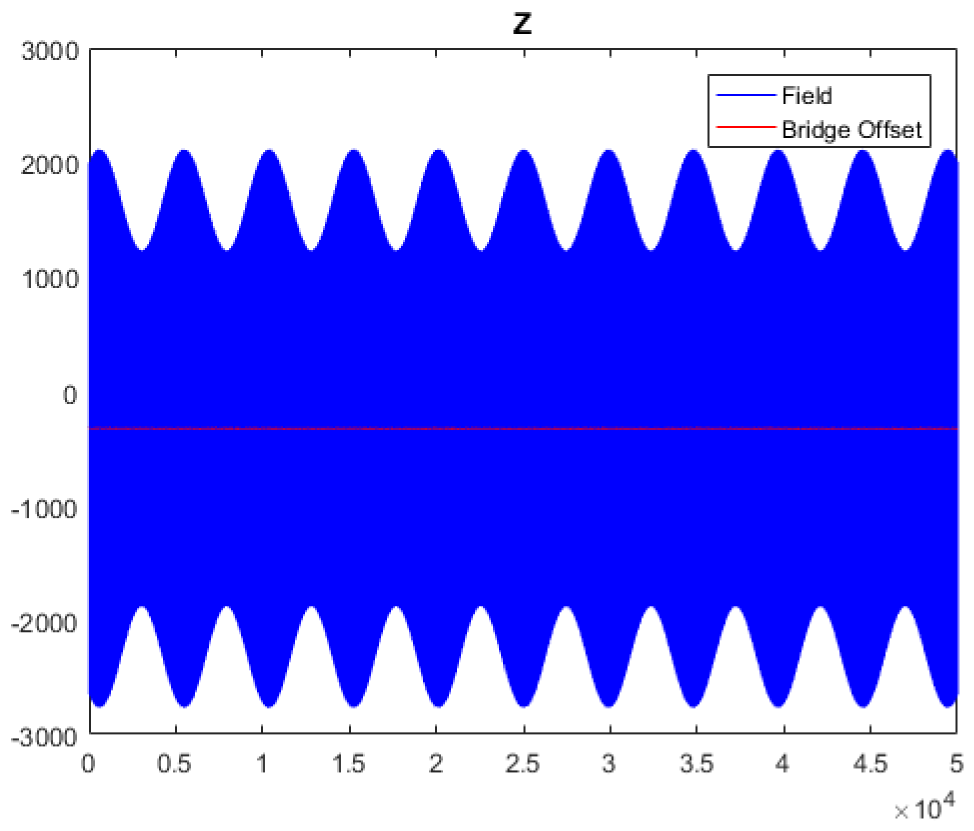


Figure 6.7: S16, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

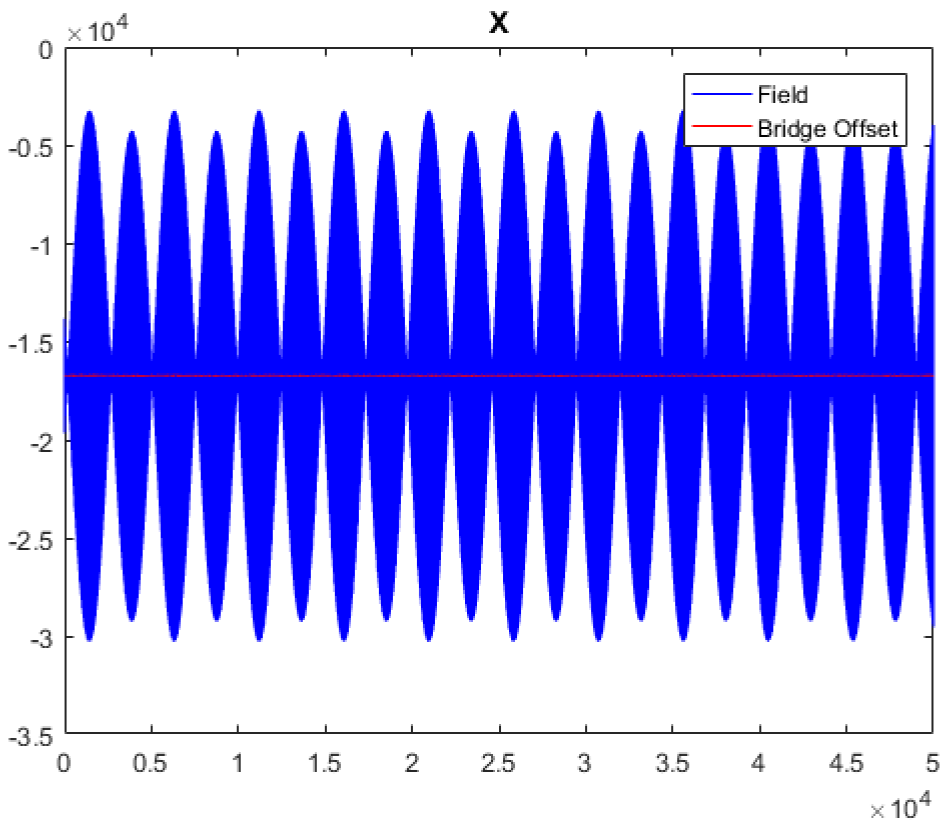


Figure 6.8: S2, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

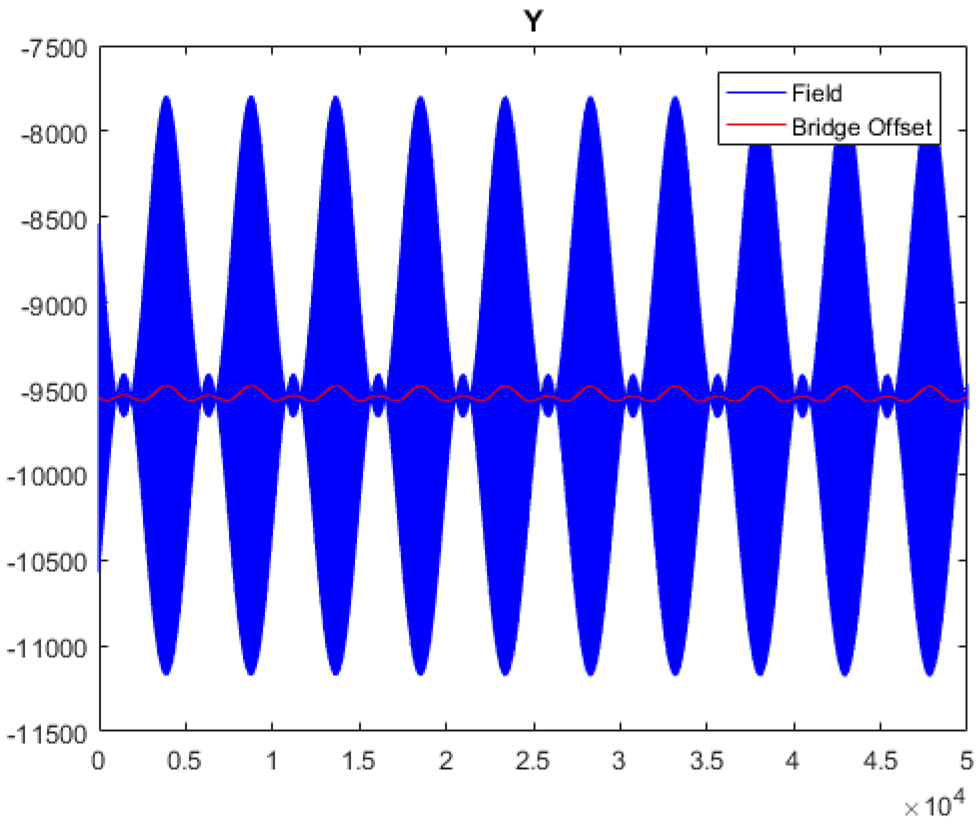


Figure 6.9: S2, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

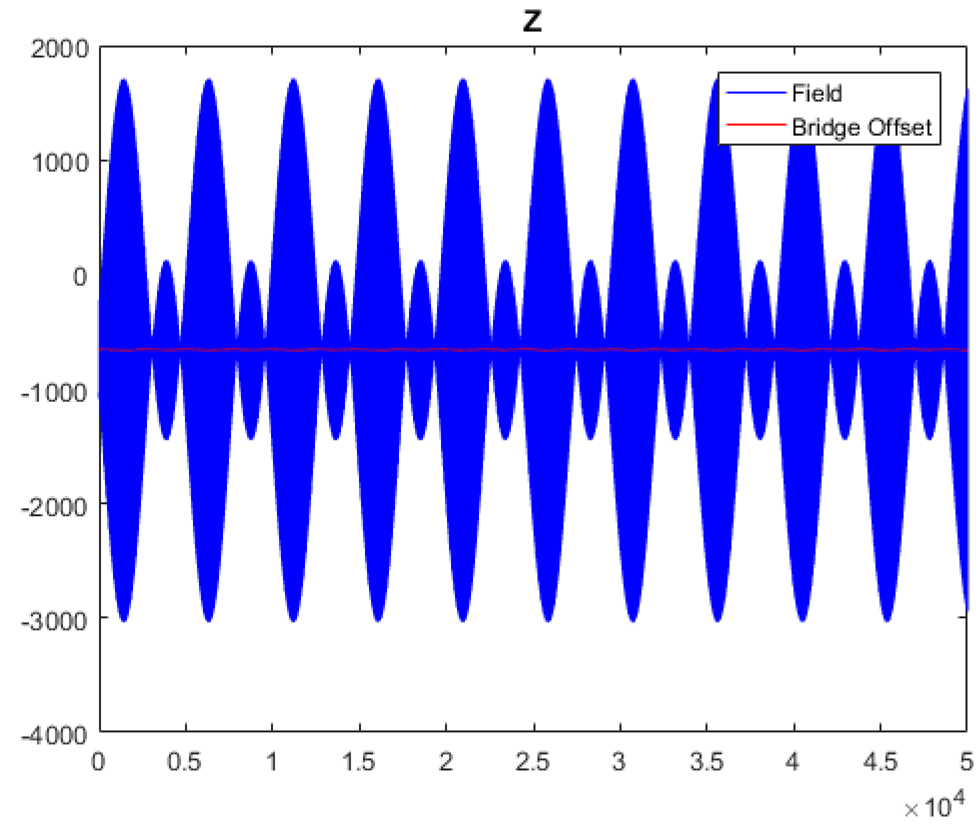


Figure 7.0: S2, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

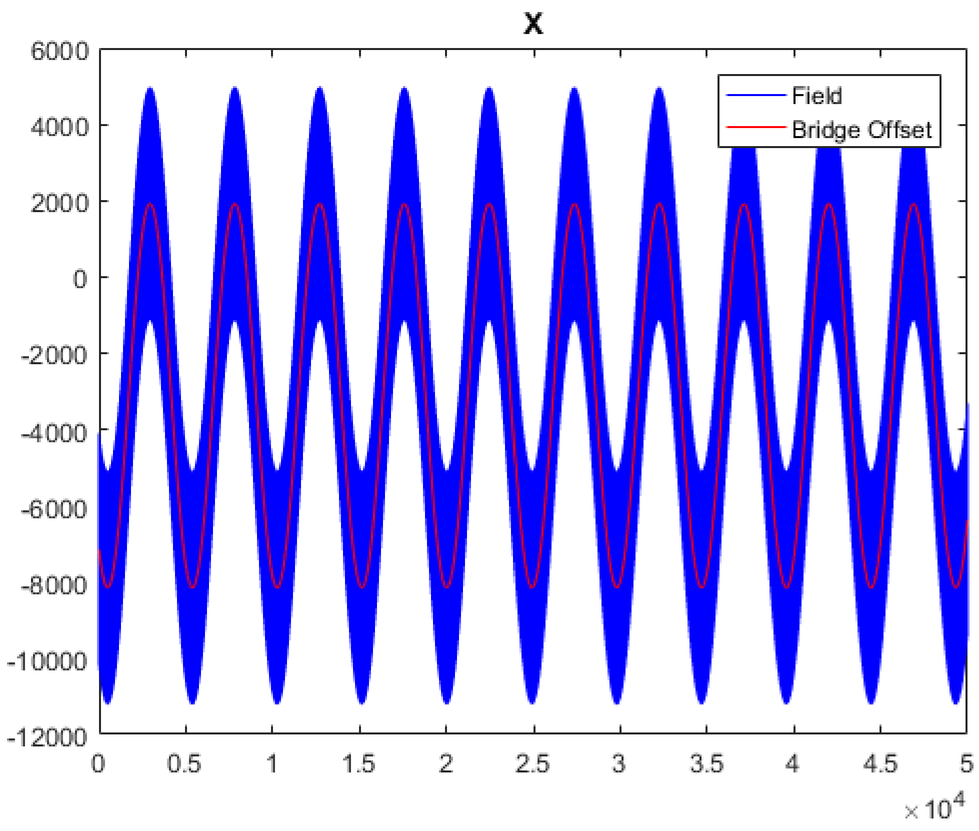


Figure 7.1: S12, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

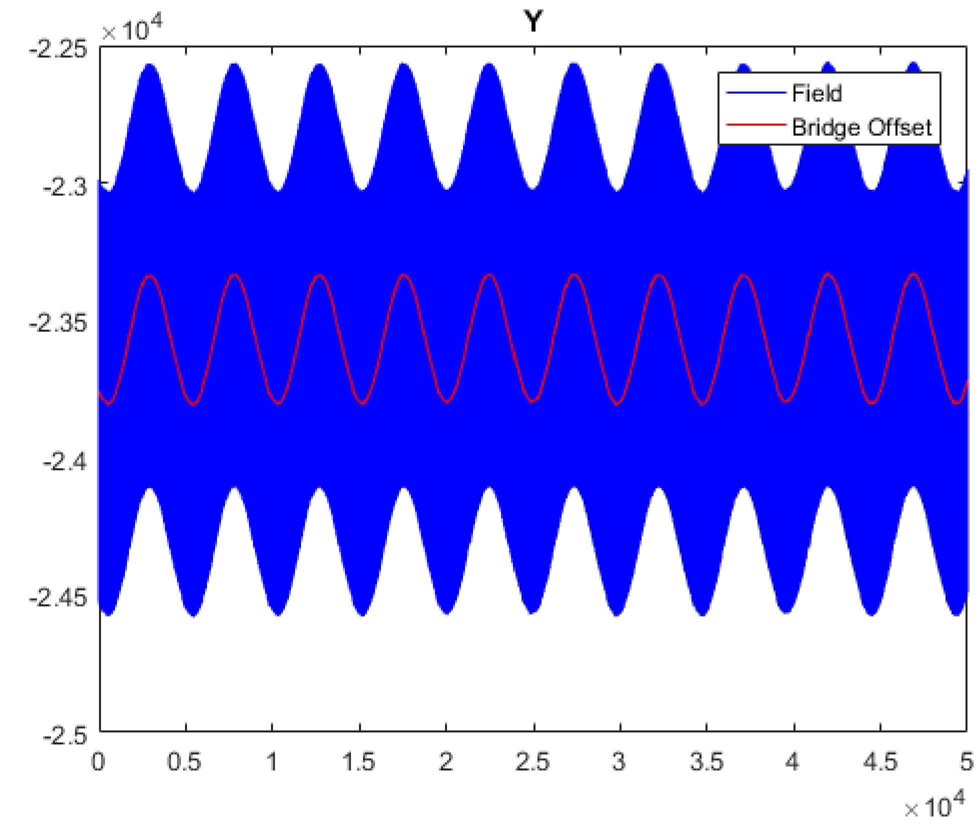


Figure 7.2: S12, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

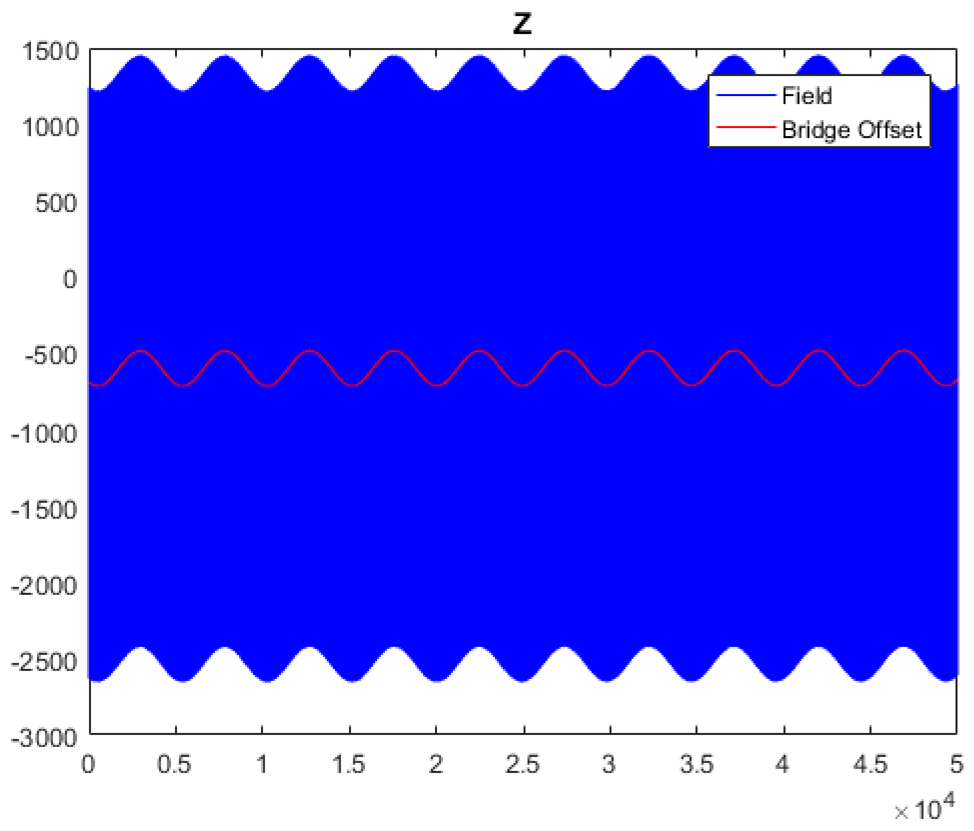


Figure 7.3: S12, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

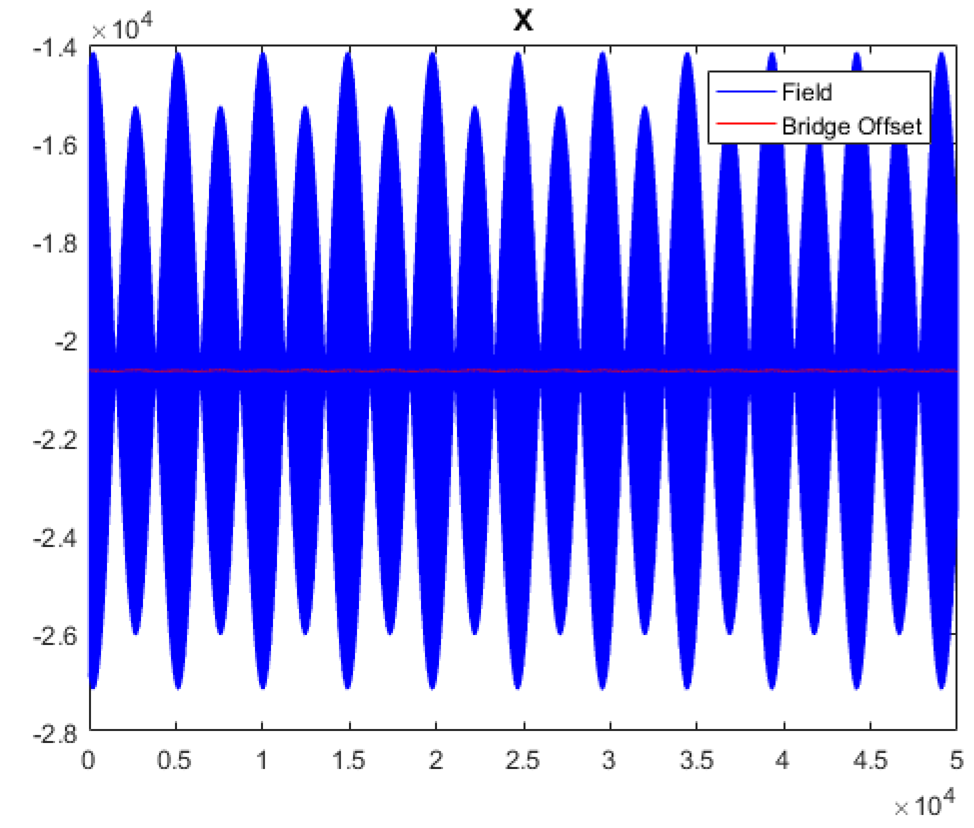


Figure 7.4: S3, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

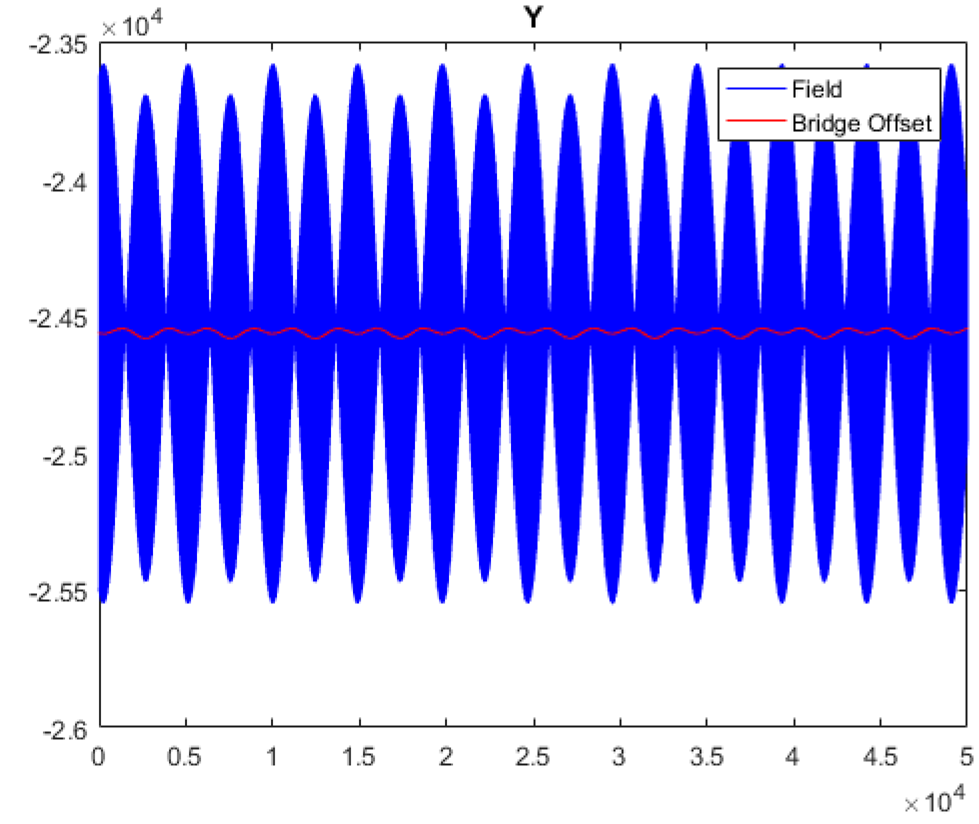


Figure 7.5: S3, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

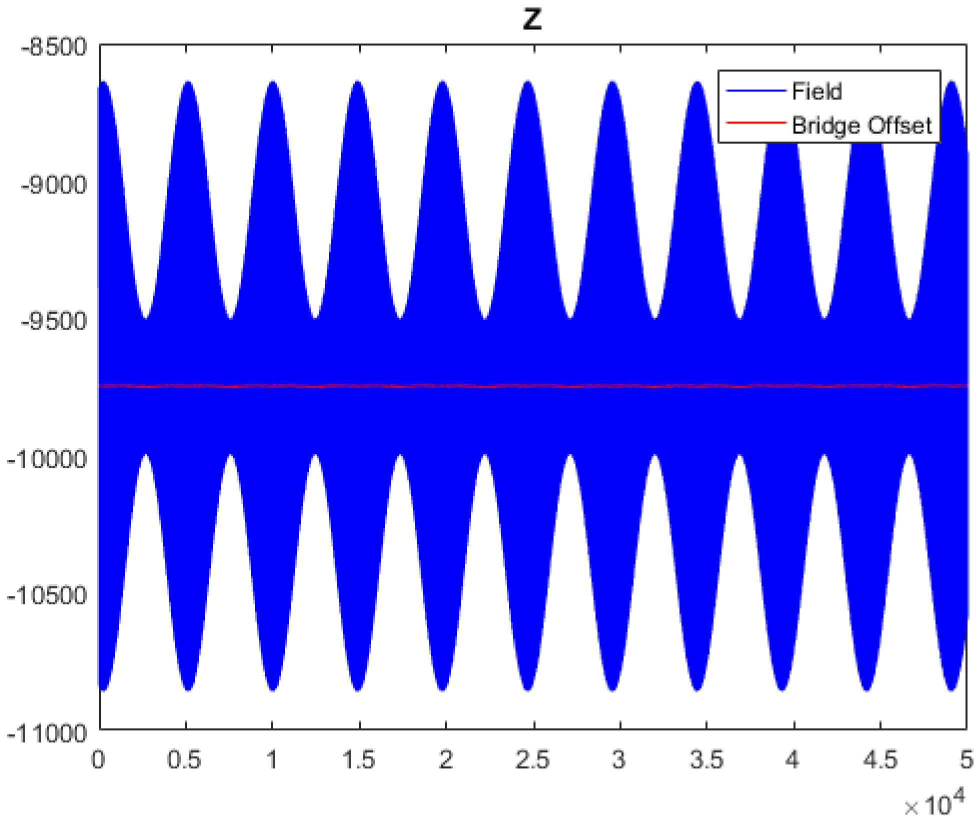


Figure 7.6: S3, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

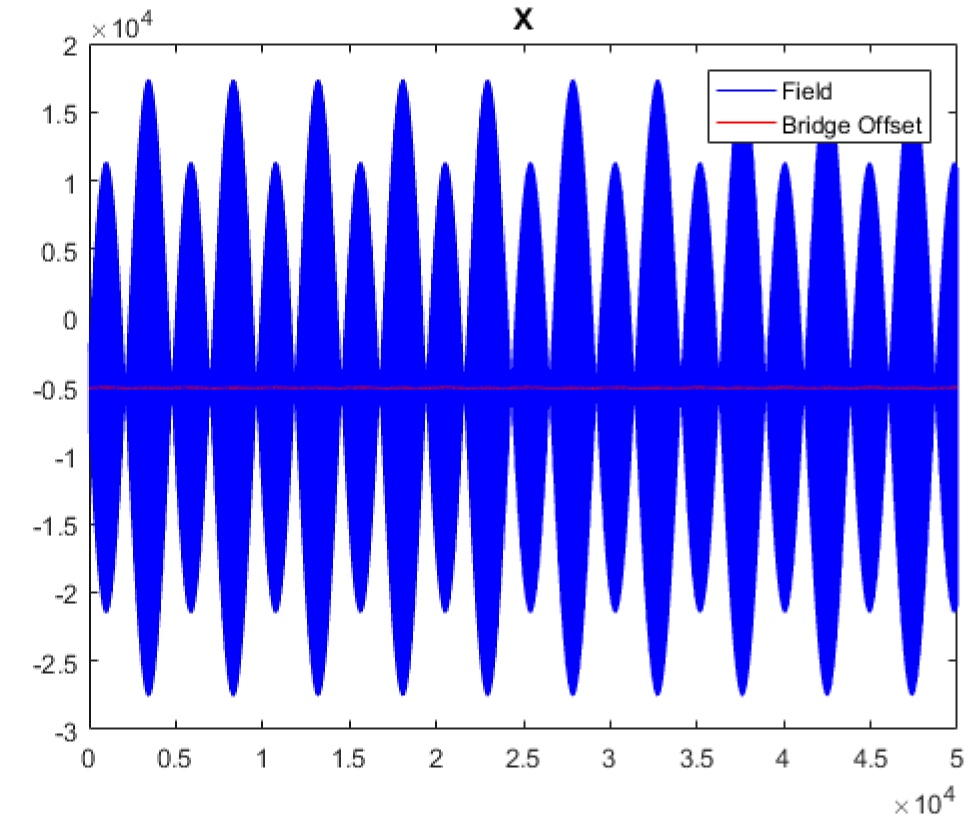


Figure 7.7: S15, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

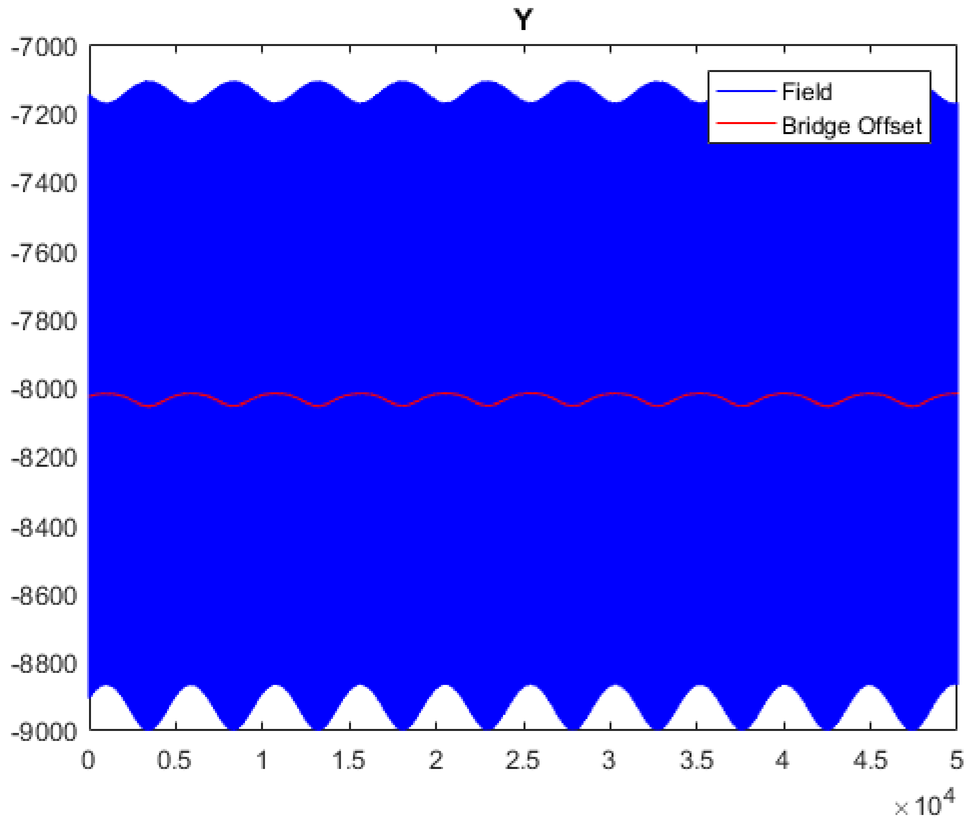


Figure 7.8 S15, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

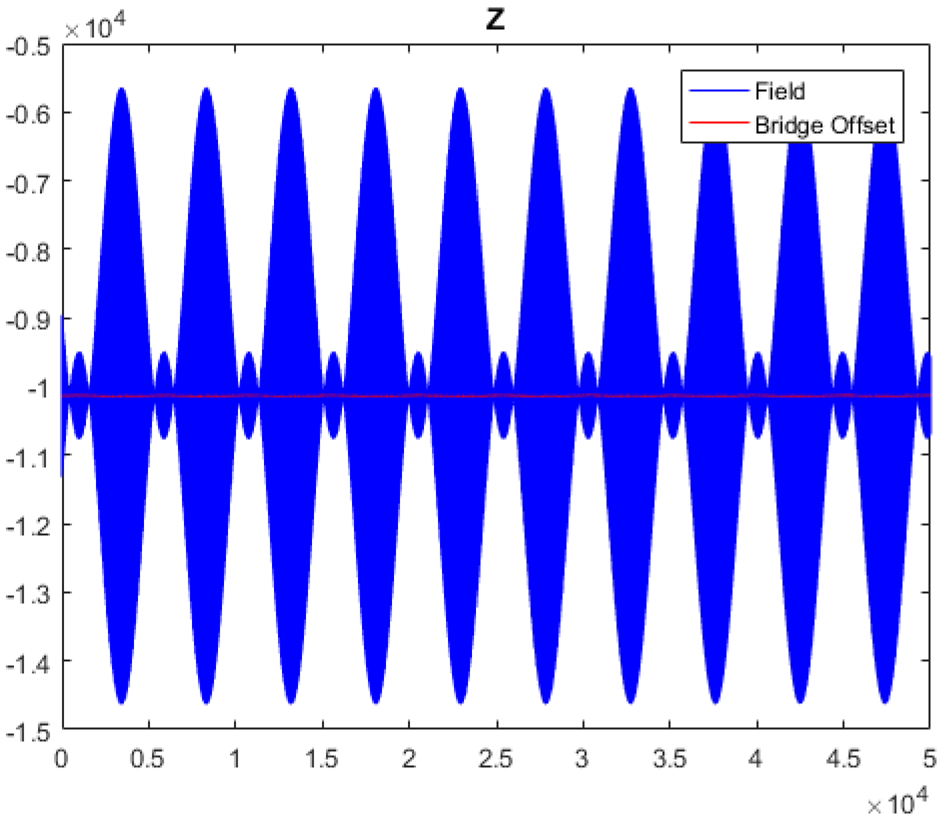


Figure 7.9: S15, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

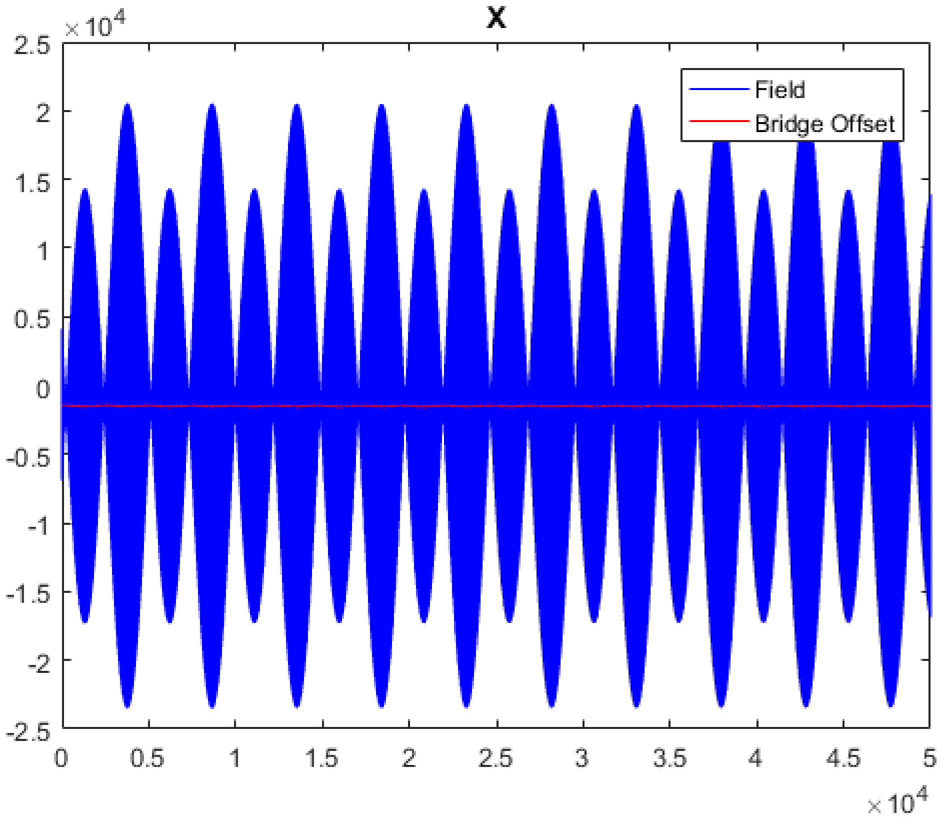


Figure 8.0: S13, X-axis in ZGC for 33000 samples of sinusoidal magnetic field

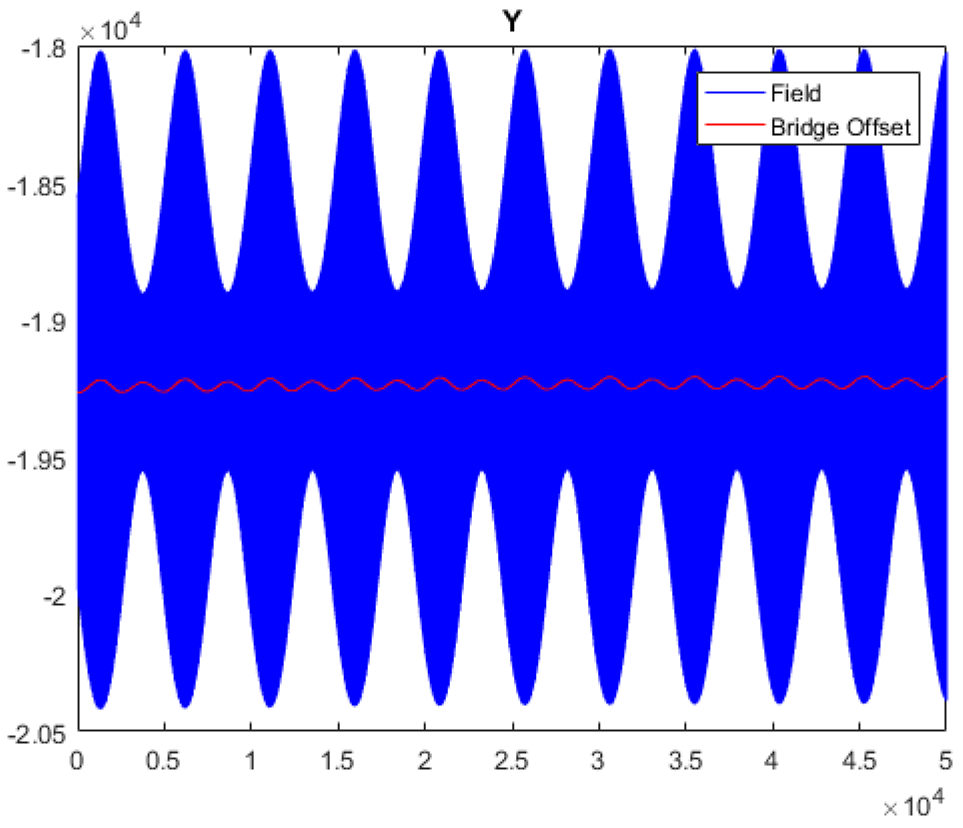


Figure 8.1: S13, Y-axis in ZGC for 33000 samples of sinusoidal magnetic field

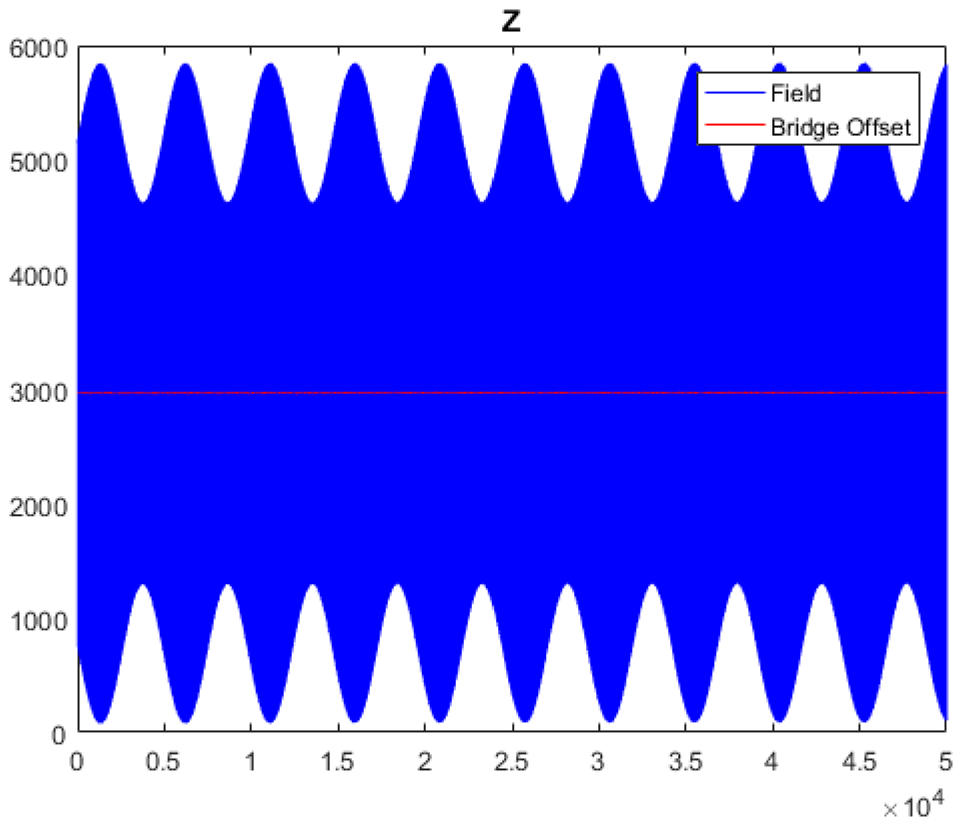


Figure 8.2: S13, Z-axis in ZGC for 33000 samples of sinusoidal magnetic field

E.3

SENSITIVITY:

```
clc
close all
clear all

defaultpaperfig

Ax=zeros(1,18);
SAx=zeros(1,18);
RSAx=zeros(1,18);
% Ay=zeros(18);
% Az=zeros(18);
[r,c]=size(load('data00000.txt'));
d=zeros(r,c,18);

n=r;

Vbx = zeros(n-1,18);
% Vby = zeros(n-1,18);
% Vbz = zeros(n-1,18);

Vax = zeros(n-1,18);
% Vay = zeros(n-1,18);
% Vaz = zeros(n-1,18);

d(:,:,1)=load('data00000.txt');
d(:,:,2)=load('data00001.txt');
d(:,:,3)=load('data00002.txt');
d(:,:,4)=load('data00003.txt');
d(:,:,5)=load('data00004.txt');
d(:,:,6)=load('data00005.txt');
d(:,:,7)=load('data00006.txt');
d(:,:,8)=load('data00007.txt');
d(:,:,9)=load('data00008.txt');
d(:,:,10)=load('data00009.txt');
d(:,:,11)=load('data00010.txt');
d(:,:,12)=load('data00011.txt');
d(:,:,13)=load('data00012.txt');
d(:,:,14)=load('data00013.txt');

x=zeros(n,18);

% y=zeros(n,18);
% z=zeros(n,18);

% Col_Start=[29,29,25,29,29,29,5,9,29];

for ii =1:14

    Col_Start=1;
    for jj=1:32
        temp=d(1,jj,ii);
        if (temp~=0)
            Col_Start=jj;
            break
        end
    end
x(:,ii)=d(:,Col_Start,ii);
```

```
% y(:,ii)=d(:,Col_Start+1,ii);
% z(:,ii)=d(:,Col_Start+2,ii);
end
Setx=zeros(r/2,14);
% Sety=zeros(r/2,1);
% Setz=zeros(r/2,1);
Resetx=zeros(r/2,14);
% Resety=zeros(r/2,1);
% Resetz=zeros(r/2,1);
for jj= 1:14
kk=1;
for ii=1:2:r-1

Setx(kk,jj)=x(ii,jj);
% Sety(kk,1)=y(ii,jj);
% Setz(kk,1)=z(ii,jj);

Resetx(kk,jj)=x(ii+1,jj);
% Resety(kk,1)=y(ii+1,jj);
% Resetz(kk,1)=z(ii+1,jj);

kk=kk+1;
end
end
% figure
% plot(Setx)
% title('Set - X ')
% xlabel('Digital level')
% ylabel('Digital level')
% figure
% plot(Resetx)
% title('Reset - X ')
% xlabel('Digital level')
% ylabel('Digital level')

for kk = 1:14
 k=1;
 for ii = 1:(n-1)/2
    ii1=ii-1;
       Vax(k,kk) = (x(2*ii+1,kk)-x(2*ii,kk))/2;
%         Vay(k,kk) = (y(2*ii+1,kk)-y(2*ii,kk))/2;
%         Vaz(k,kk) = (z(2*ii+1,kk)-z(2*ii,kk))/2;

       Vbx(k,kk) = (x(2*ii+1,kk)+x(2*ii,kk))/2;
%         Vby(k,kk) = (y(2*ii+1,kk)+y(2*ii,kk))/2;
%         Vbz(k,kk) = (z(2*ii+1,kk)+z(2*ii,kk))/2;
      k=k+1;

       Vax(k,kk) = (x(2*ii1+1,kk)-x(2*ii,kk))/2;
%         Vay(k,kk) = (y(2*ii1+1,kk)-y(2*ii,kk))/2;
%         Vaz(k,kk) = (z(2*ii1+1,kk)-z(2*ii,kk))/2;

       Vbx(k,kk) = (x(2*ii1+1,kk)+x(2*ii,kk))/2;
%         Vby(k,kk) = (y(2*ii1+1,kk)+y(2*ii,kk))/2;
%         Vbz(k,kk) = (z(2*ii1+1,kk)+z(2*ii,kk))/2;
      k=k+1;
 end
end
Rx=zeros(18,1);
S_Rx=zeros(18,1);
RS_Rx=zeros(18,1);
```

```
for ii = 1:14
 label = sprintf('%01d',ii);
 label1 = ['set',sprintf('%01d',ii)];
 label2 = ['reset',sprintf('%01d',ii)];
n=50000;
figure
% plot(x(:,ii))
% hold on
plot(Vax(:,ii))
hold on
fitx=ezfit(Vax(:,ii),'a*cos(2*pi*f*x+p)+b;f=2e-4;a=16000;p=0.1;b=-3000');
showfit(fitx)
makevarfit(fitx)
Ax(1,ii)=a;
Rx(ii,1)=fitx.r;
title(strcat('X - ',num2str(ii-1)))
    xlabel('digital level')
    ylabel('digital level')
print([label],'-dpng')

figure
% plot(x(:,ii))
% hold on
plot(Setx(:,ii))
hold on
fitx=ezfit(Setx(:,ii),'a*cos(2*pi*f*x+p)+b;f=4e-4;a=30000;p=0.1;b=-3000');
showfit(fitx)
makevarfit(fitx)
SAx(1,ii)=a;
S_Rx(ii,1)=fitx.r;
title(strcat('X - set ',num2str(ii-1)))
    xlabel('digital level')
    ylabel('digital level')
   print(label1,'-dpng')

figure
% plot(x(:,ii))
% hold on
plot(Resetx(:,ii))
hold on
fitx=ezfit(Resetx(:,ii),'a*cos(2*pi*f*x+p)+b;f=4e-4;a=30000;p=0.1;b=-
3000');
showfit(fitx)
makevarfit(fitx)
RSAx(1,ii)=a;
RS_Rx(ii,1)=fitx.r;
title(strcat('X - reset',num2str(ii-1)))
    xlabel('digital level')
    ylabel('digital level')
    print(label2,'-dpng')
% figure
% plot(y(:,ii))
% hold on
% plot(Vay(:,ii))
% hold on
% fity=ezfit(Vay(:,ii),'a*cos(2*pi*f*x+p)+b;f=2e-4;a=62;p=0.1;b=-800');
% showfit(fity)
% makevarfit
% a
% Ay(ii)=a;
% title(strcat('Y - ',num2str(ii-1)))
% figure
% plot(z(:,ii))
% hold on
% plot(Vaz(:,ii))
% hold on
% fitz=ezfit(Vaz(:,ii),'a*cos(2*pi*f*x+p)+b;f=2e-4;a=1100;p=0.1;b=-2000');
% showfit(fitz)
% makevarfit
% a
% Az(ii)=a;
% title(strcat('Z - ',num2str(ii-1)))
end
Vapp=[6,5,5,4,6,4,6,5,5,4,2,2,6,6,1,1,1,1];%applied voltage
Ax_per_Vapp=zeros(1,18);
figure
for ii=1:14
    Ax_per_Vapp(1,ii)=Ax(1,ii)/Vapp(1,ii);

end
plot(Ax/10^4,'r-')
hold on
plot(Vapp,'b-')
hold on
plot(Ax_per_Vapp/10^4,'k*')
title('Ampitidue - voltage dependency')
legend('Amplitude/exp 4','Volts applied','Amplitude/exp 4 per Volts
applied')
xlabel('sensors')
ylabel('digital/exp 4, Volts')
print('Ampitidue voltage dependency ','-dpng')
% figure
% plot(Ay,'-')
% title('Y Max-Amplitudes')
% figure
% plot(Az,'-')
% title('Z Max-Amplitudes')

B=[1.403,1.1775,1.1775,0.9301,1.4155,0.9395,1.403,1.2026,1.1775,0.9269,0.48
85,0.4885,1.4155,1.4155];%in G
Vppcx=2*abs(Ax);%Vpp of coil in x direction

S_Vppcx=2*abs(SAx);
RS_Vppcx=2*abs(RSAx);

Sensx=zeros(18,1);

S_Sensx=zeros(18,1);
RS_Sensx=zeros(18,1);

for ii=1:14
    Sensx(ii)=B(ii)*(10^6)/Vppcx(ii);
    S_Sensx(ii)=B(ii)*(10^6)/S_Vppcx(ii);
    RS_Sensx(ii)=B(ii)*(10^6)/RS_Vppcx(ii);
%     Sensy(ii)=B(ii)/Vppcy(ii);
%     Sensz(ii)=B(ii)/Vppcz(ii);
end
figure
plot(Sensx)
hold on
```

```
% title('X Sensitivity in Micro Gauss per digital level')
% xlabel('sensors')
% ylabel('micro gauss/digital level')
%
%
% figure
plot(S_Sensx)
hold on
%title('X Set-Sensitivity in Micro Gauss per digital level')
% xlabel('sensors')
% ylabel('micro gauss/digital level')
%
% figure
plot(RS_Sensx)
title('X Sensitivity in Micro Gauss per digital level')
xlabel('sensors')
ylabel('micro gauss/digital level')
legend('Sensitivity','Set - Sensitivity',' Reset - Sensitivity')
print('X Sensitivity in Micro Gauss per digital level','-dpng')
figure
plot(Rx)
hold on
plot(S_Rx)
hold on
plot(RS_Rx)
xlabel('sensor')
ylabel('Linear regression')
title('regression')
legend('linear regression','set linear regression','reset linear
regression')
print('regression ','-dpng')
% figure
% plot(Sensy)
% title('Y Sensitivity in Gauss per unit')
% figure
% plot(Sensz)
% title('Z Sensitivity in Gauss per unit')
GS_Sensx = zeros(8,1);
jj=1;
for ii = 1:14
     if(ii~= 3 && ii~= 8 && ii~=11 && ii~=14 && ii~=2 && ii~=12)
         GS_Sensx(jj)=Sensx(ii,1);
         jj=jj+1;
     end
end
figure
plot(GS_Sensx)
hold on
Sen_mean=zeros(8,1)
for ii=1:8
Sen_mean(ii)=mean(GS_Sensx)
end
plot(Sen_mean,'r')
title('sensitivity of good sensors in micro gauss per digital level')
legend('Sensitivity','Mean')
xlabel('Sensors')
ylabel('Micro gauss/digital level')
print('Sensitivity of good sensor ','-dpng')
save ('Sensitivity','Ax','B','Vppcx','Sensx');
```

E.4

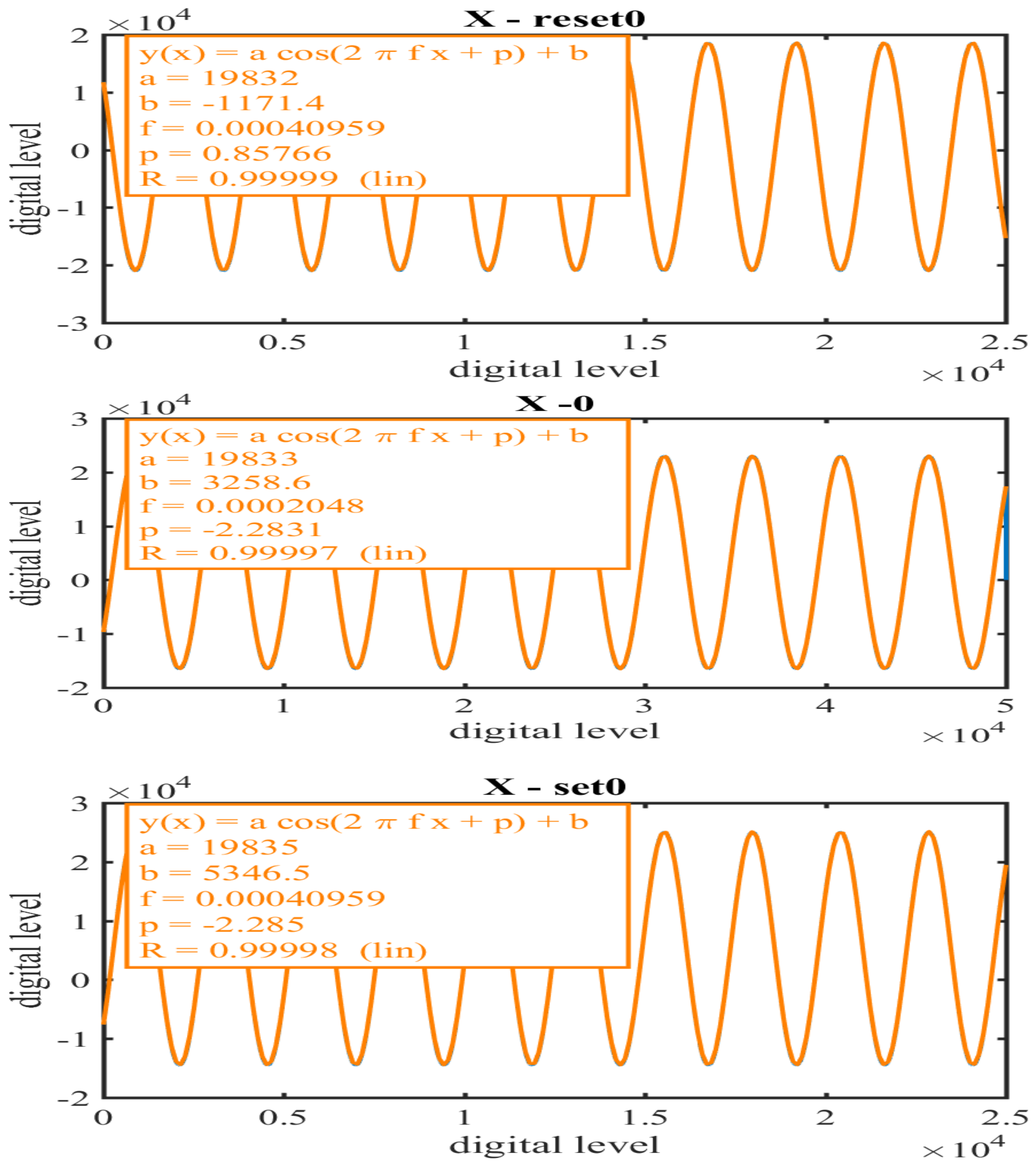


Figure 8.3: Amplitudes – Overall, Set, and Reset for data00000.txt

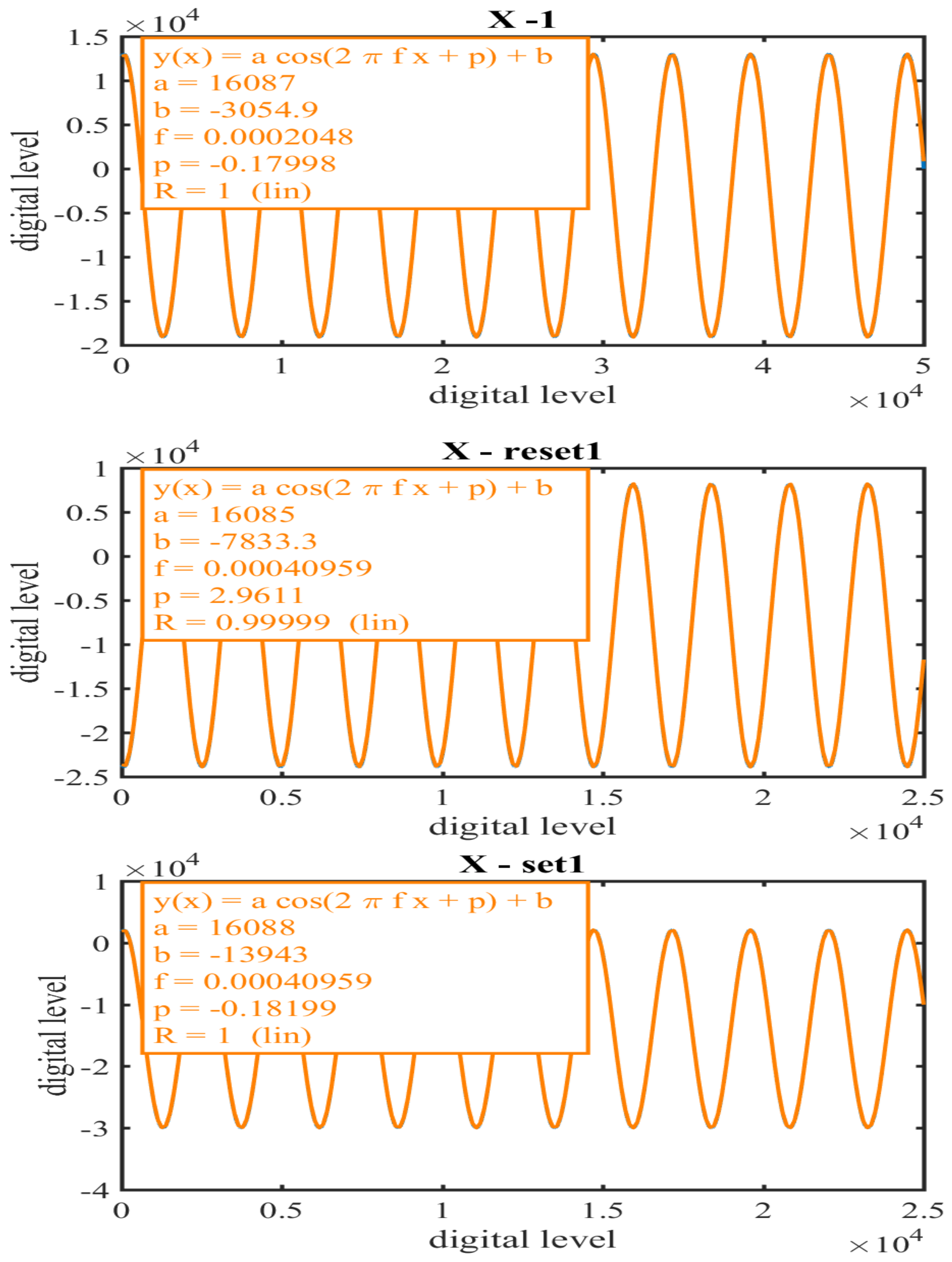


Figure 8.4: Amplitudes – Overall, Set, and Reset for data00001.txt

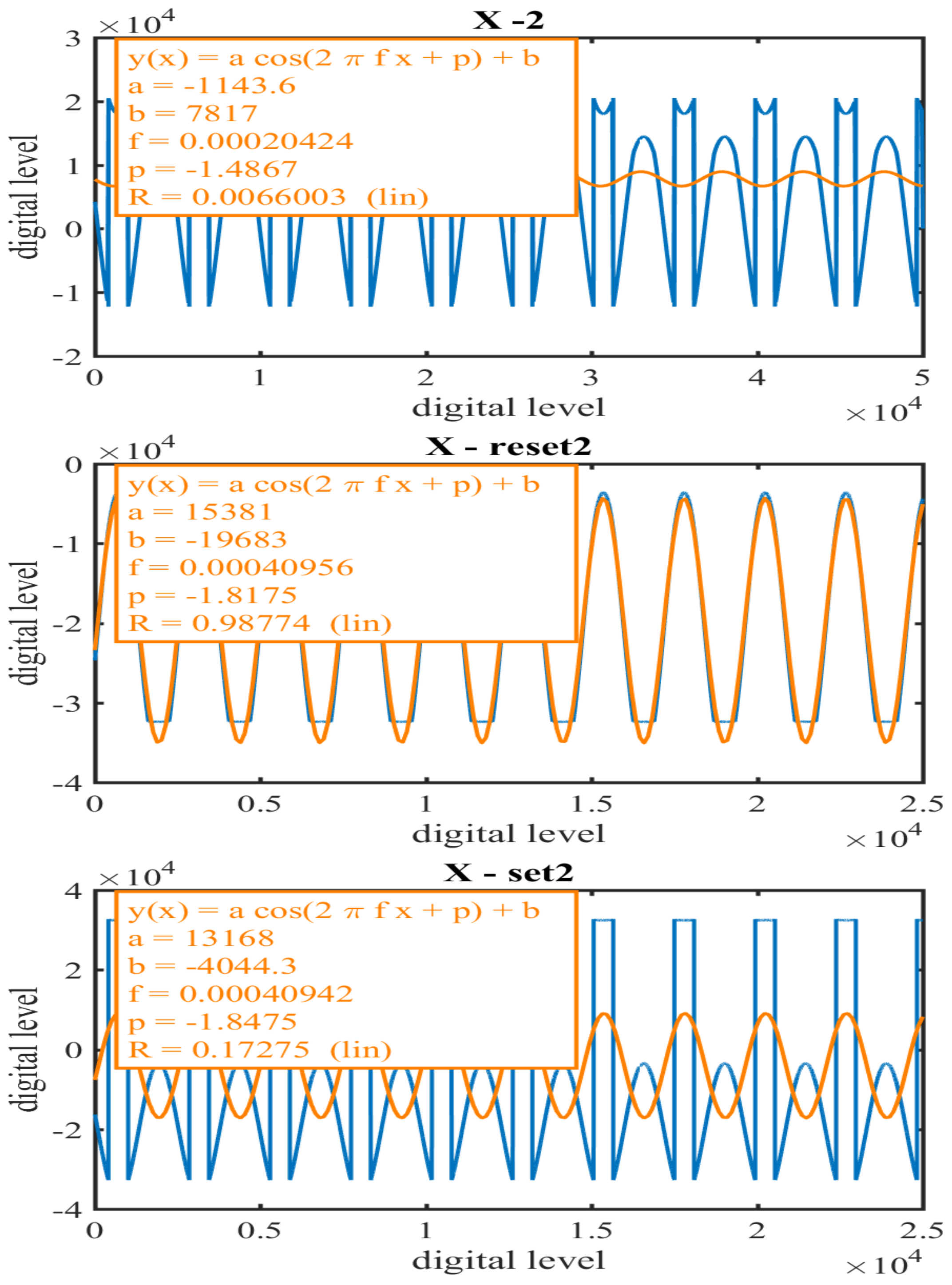


Figure 8.5: Amplitudes – Overall, Set, and Reset for data00002.txt

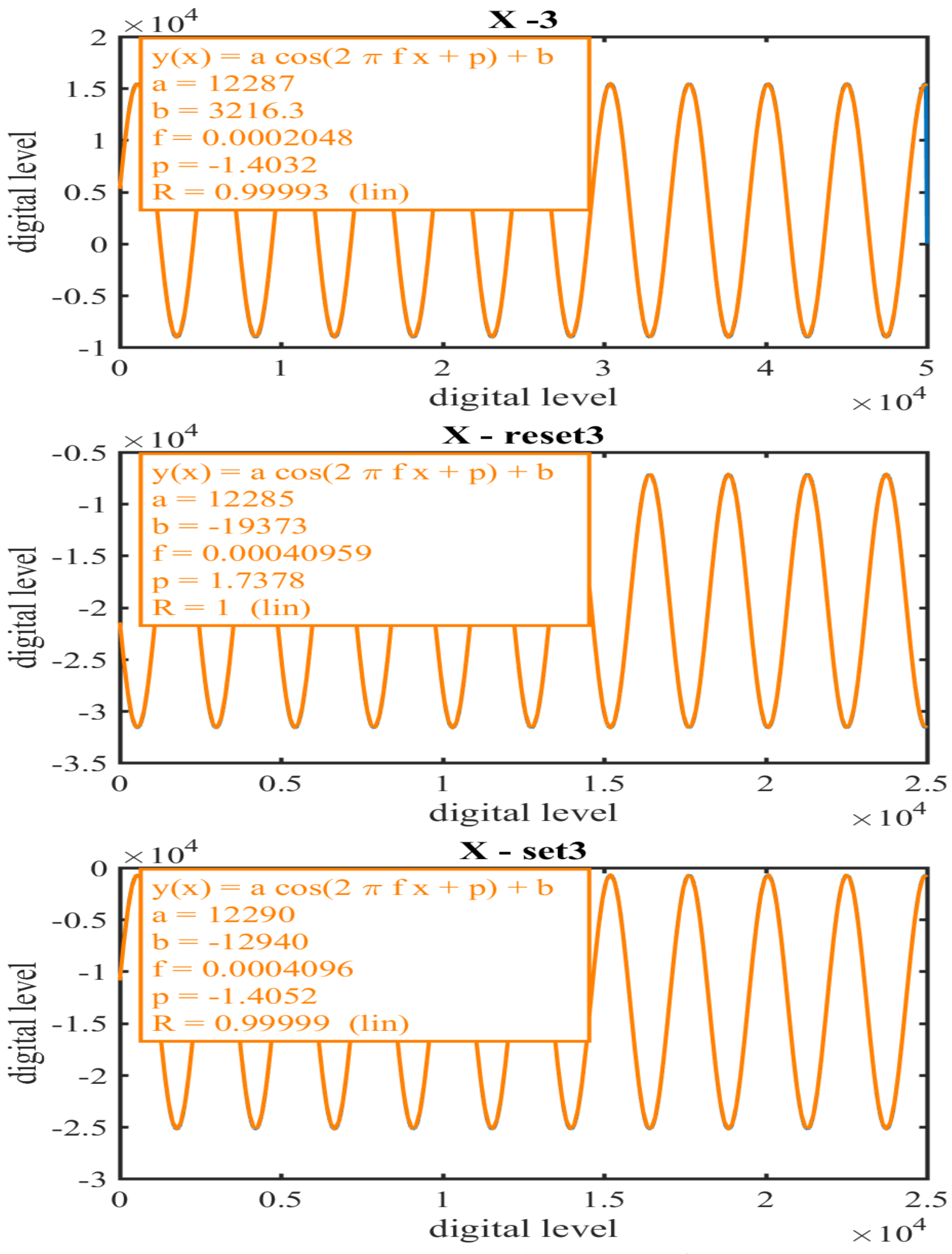


Figure 8.6: Amplitudes – Overall, Set, and Reset for data00003.txt

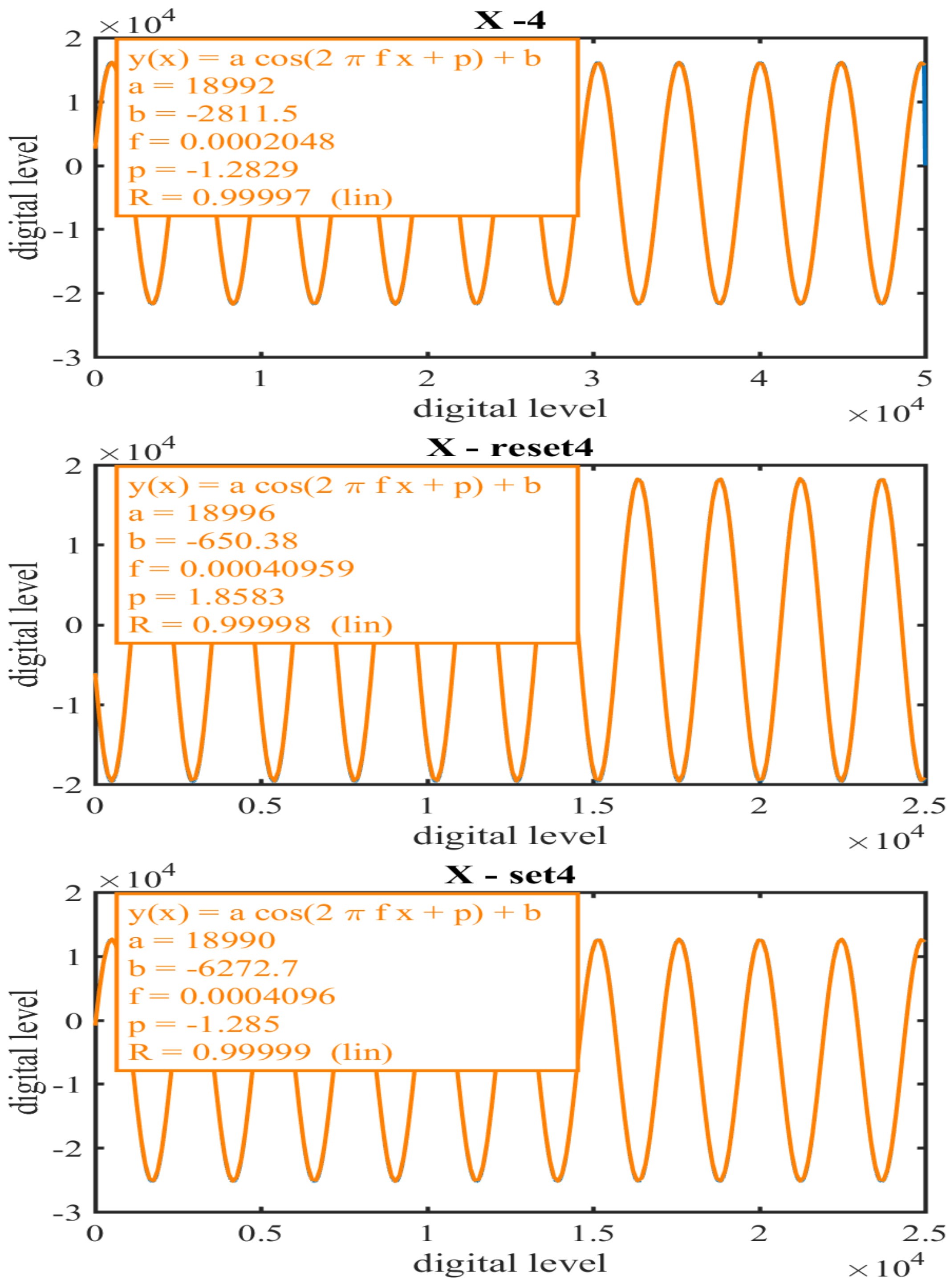


Figure 8.7: Amplitudes – Overall, Set, and Reset for data00004.txt

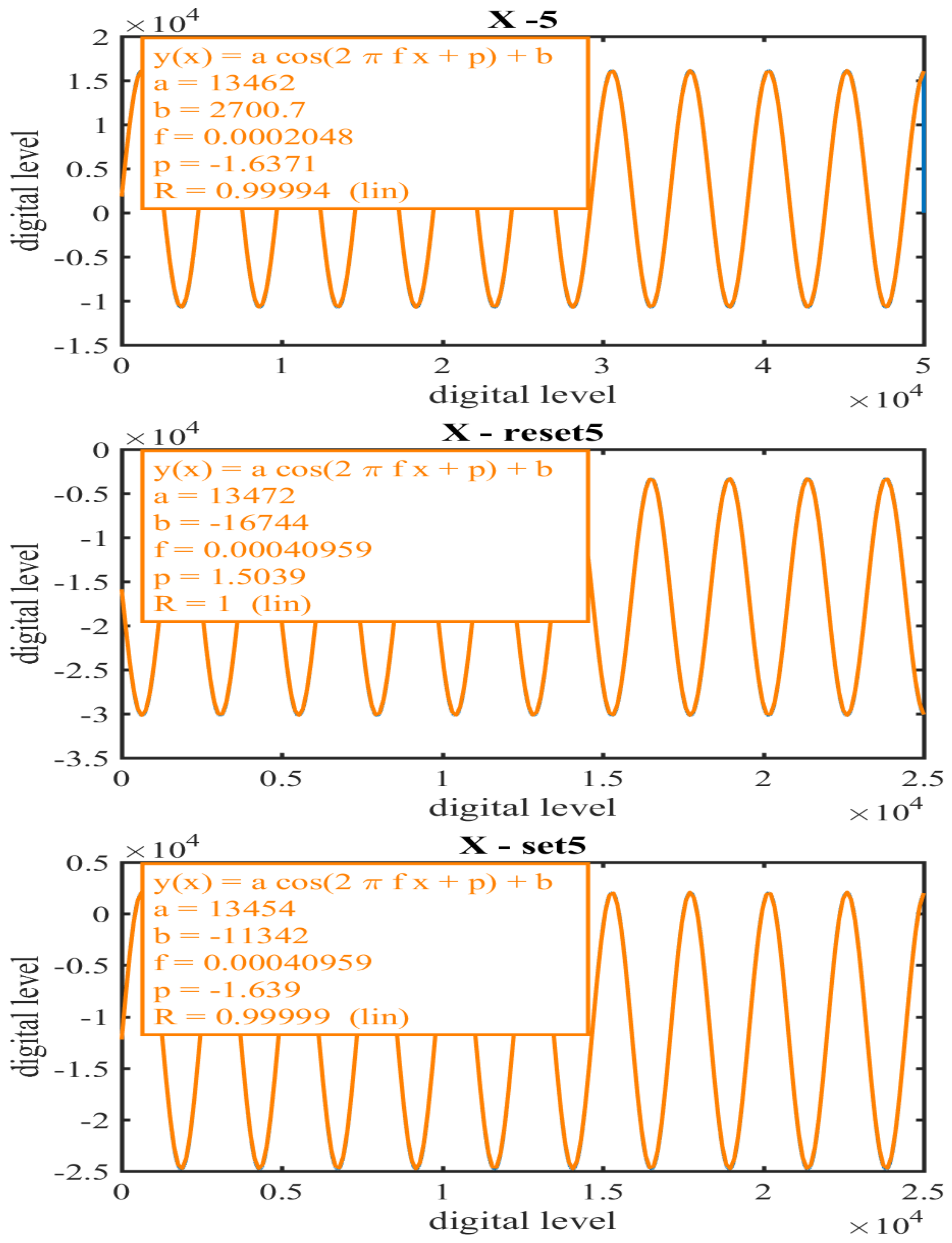


Figure 8.8: Amplitudes – Overall, Set, and Reset for data00005.txt

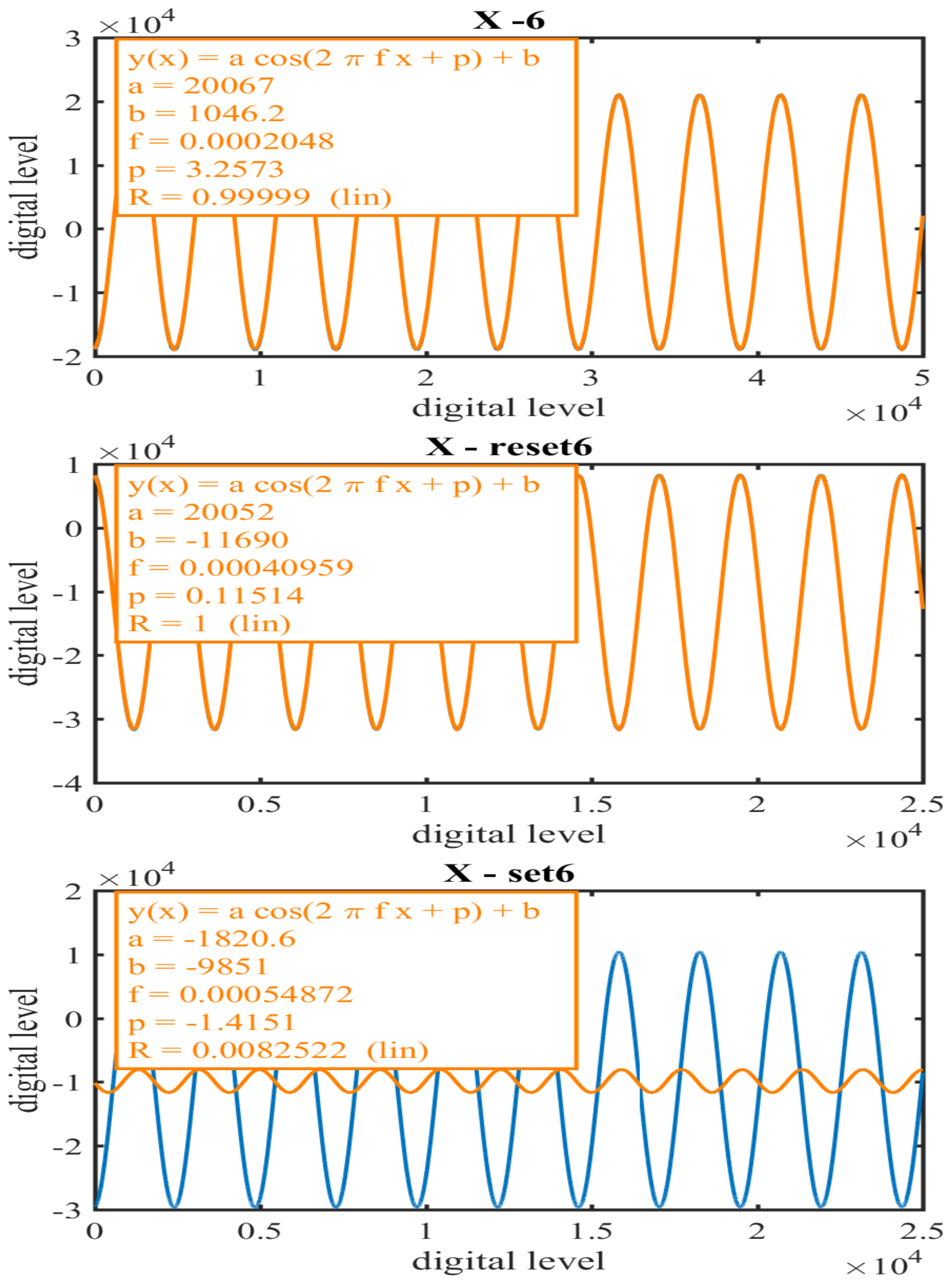


Figure 8.9: Amplitudes – Overall, Set, and Reset for data00006.txt

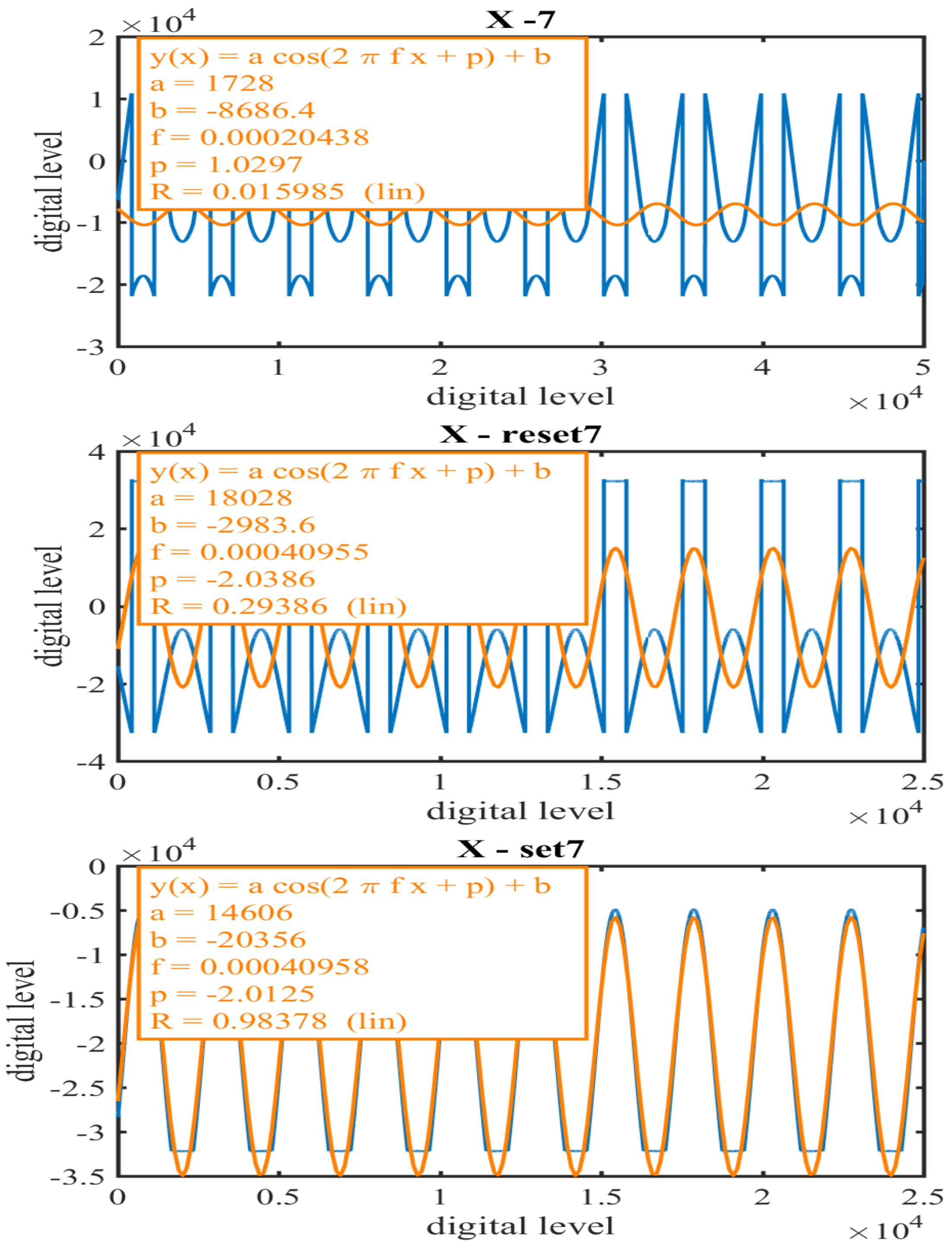


Figure 9.0: Amplitudes – Overall, Set, and Reset for data00007.txt

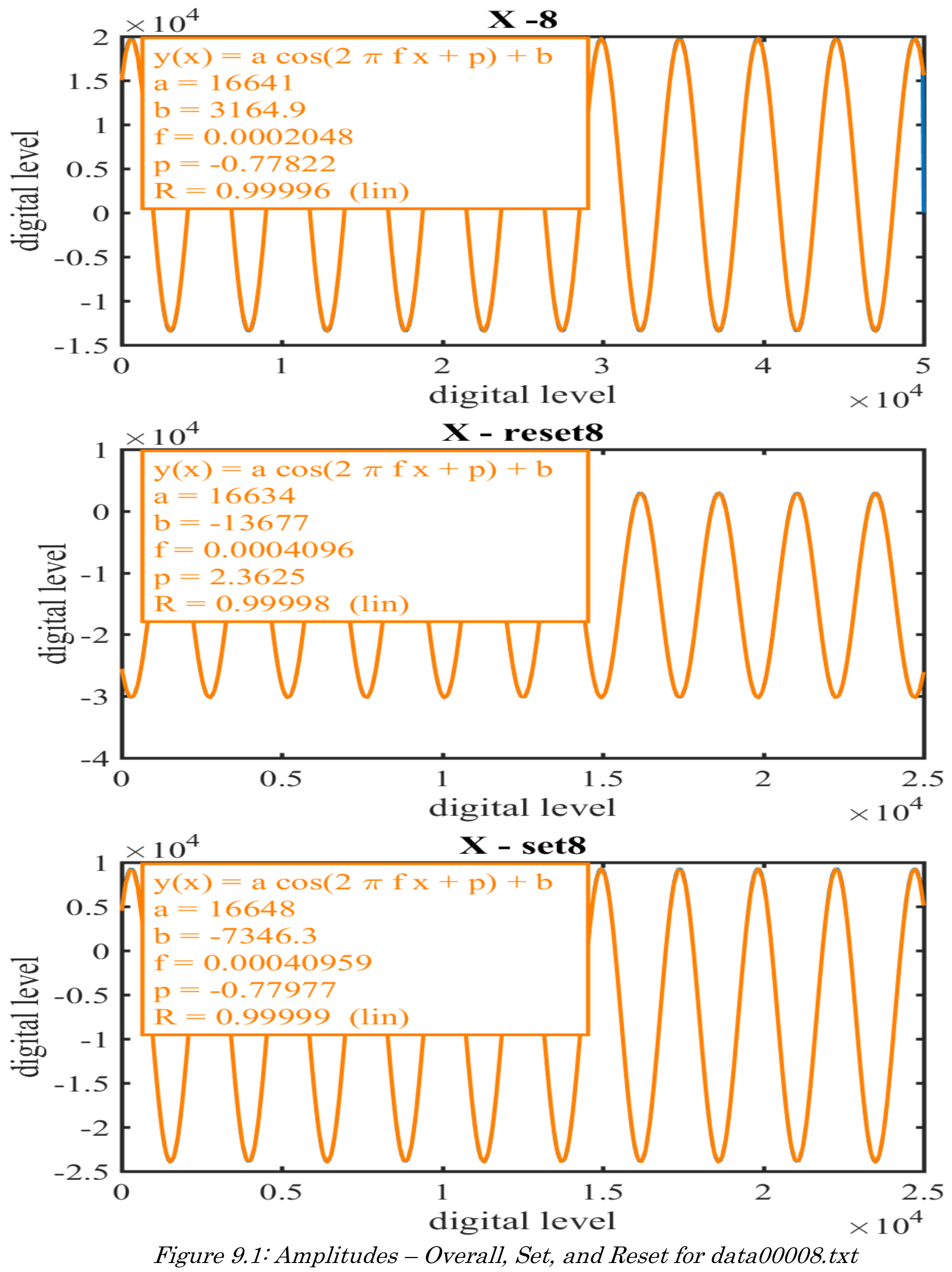


Figure 9.1: Amplitudes – Overall, Set, and Reset for data00008.txt

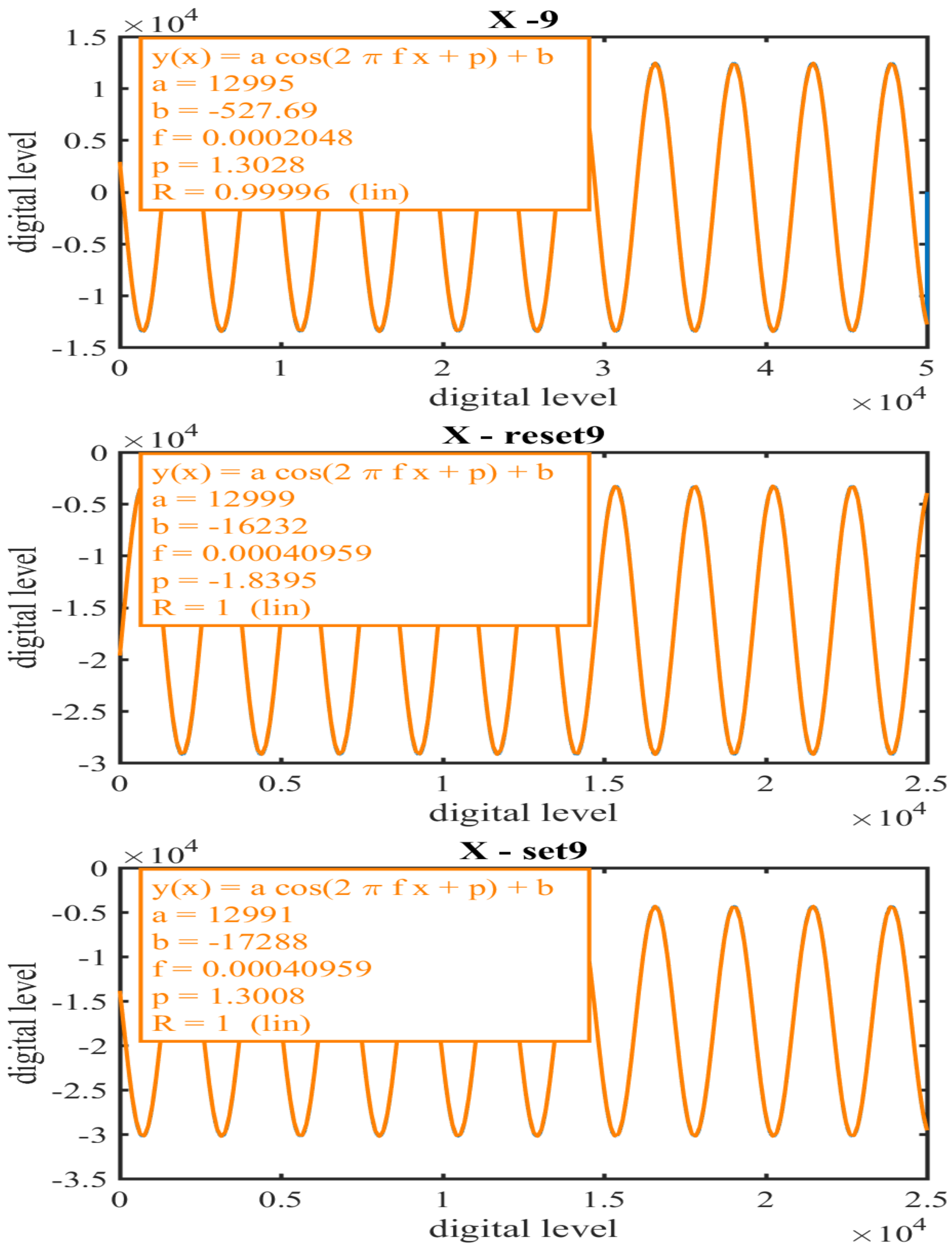


Figure 9.2: Amplitudes – Overall, Set, and Reset for data00009.txt

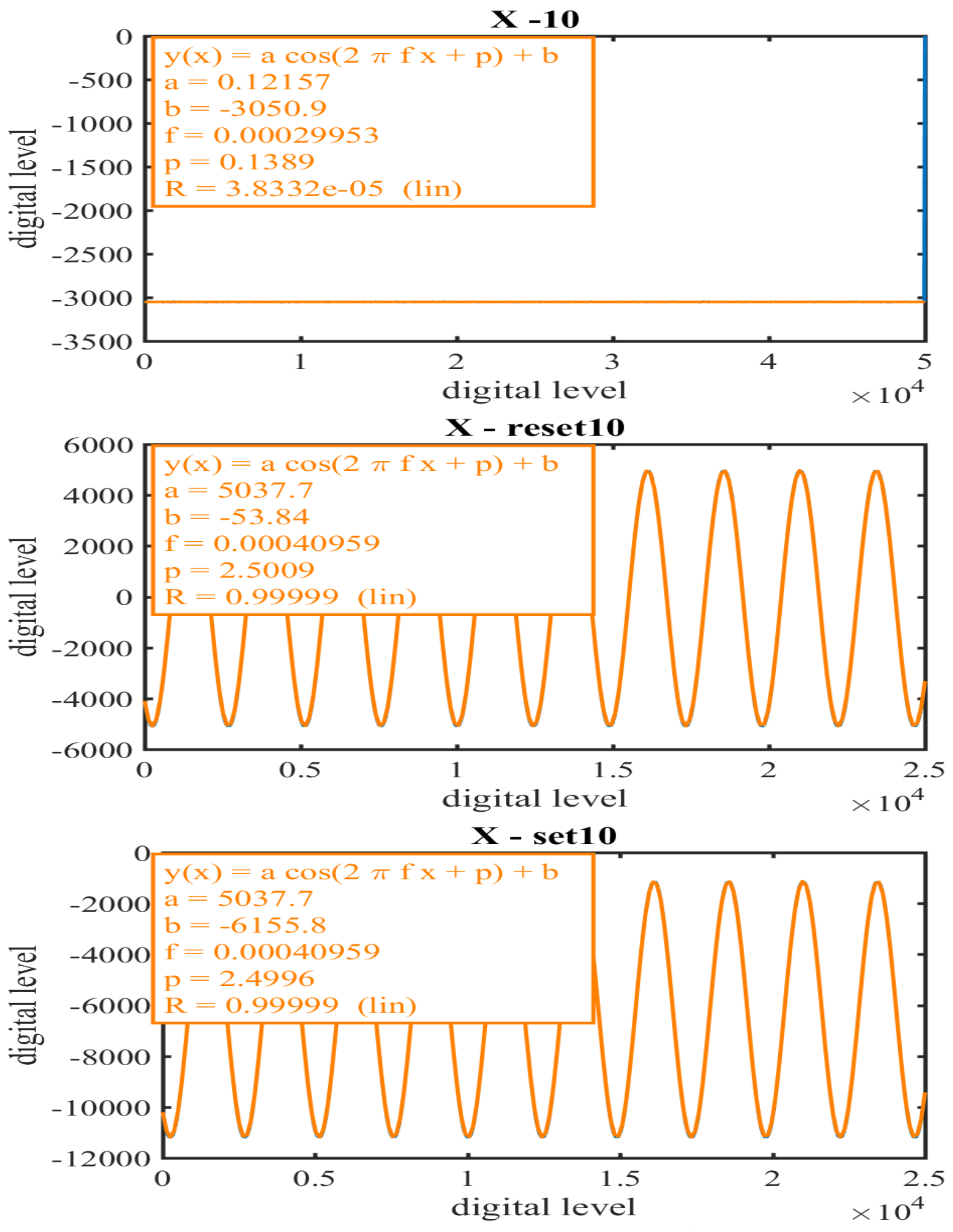


Figure 9.3: Amplitudes – Overall, Set, and Reset for data00010.txt

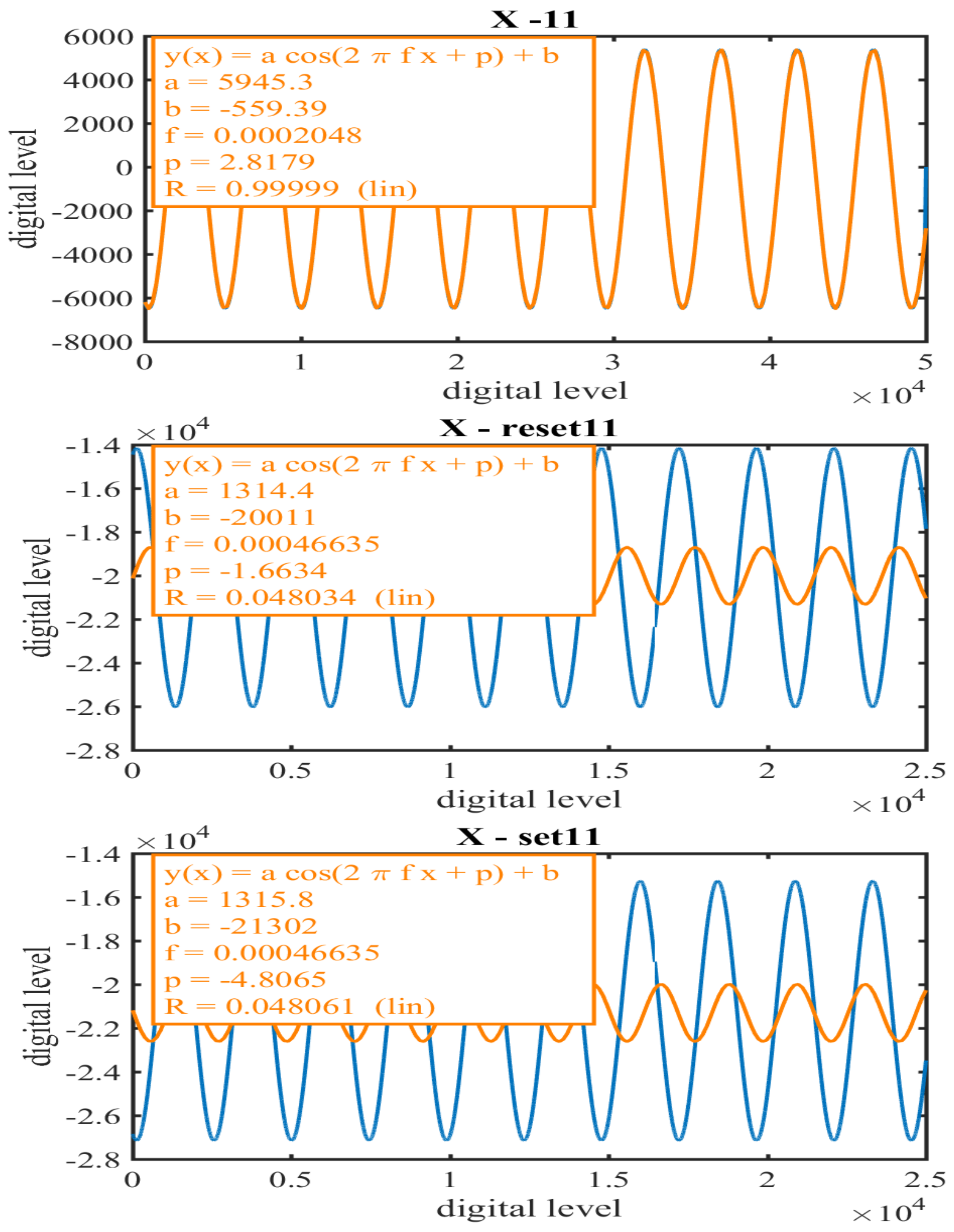


Figure 9.4: Amplitudes – Overall, Set, and Reset for data00011.txt

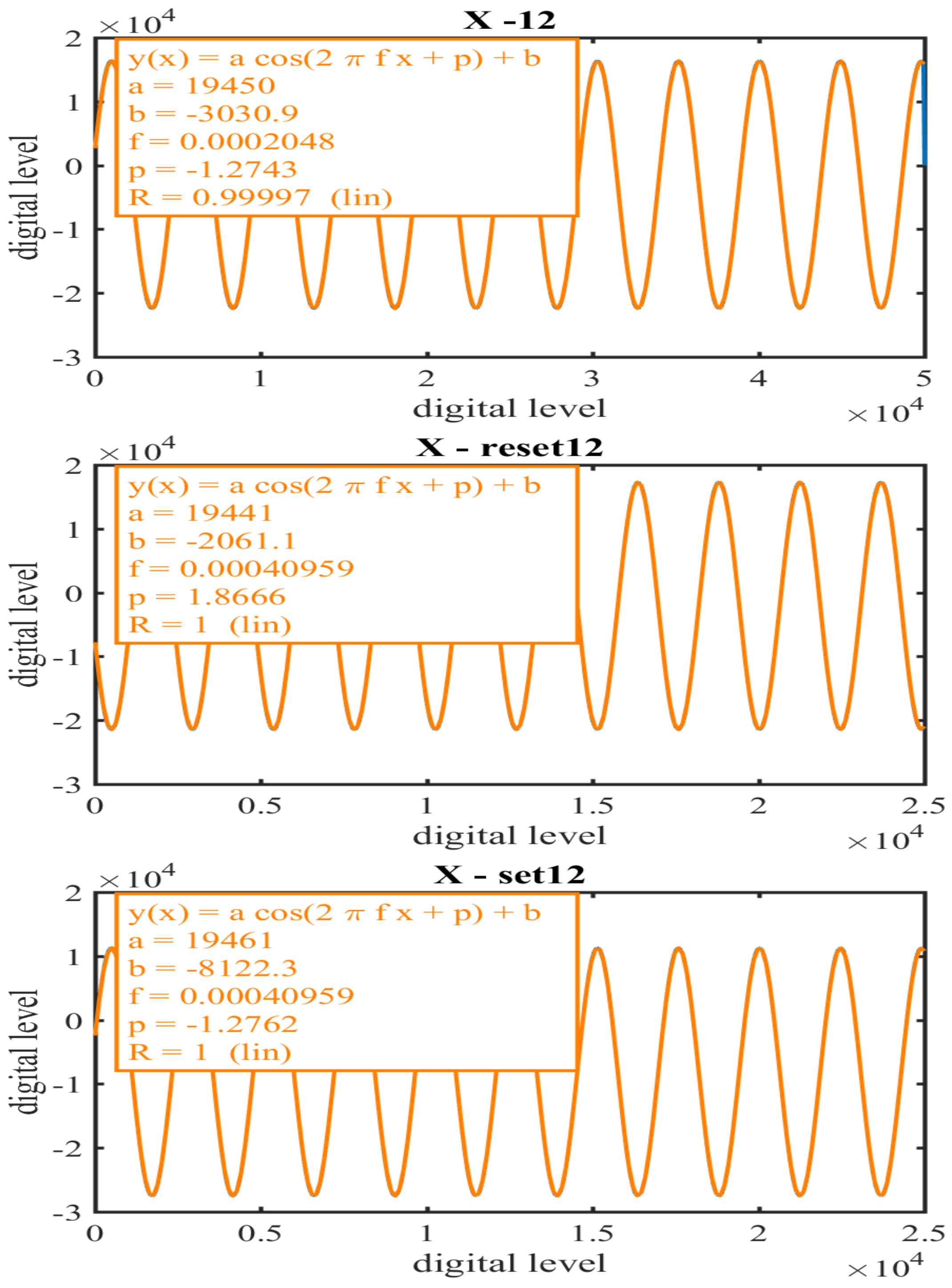


Figure 9.5: Amplitudes – Overall, Set, and Reset for data00012.txt

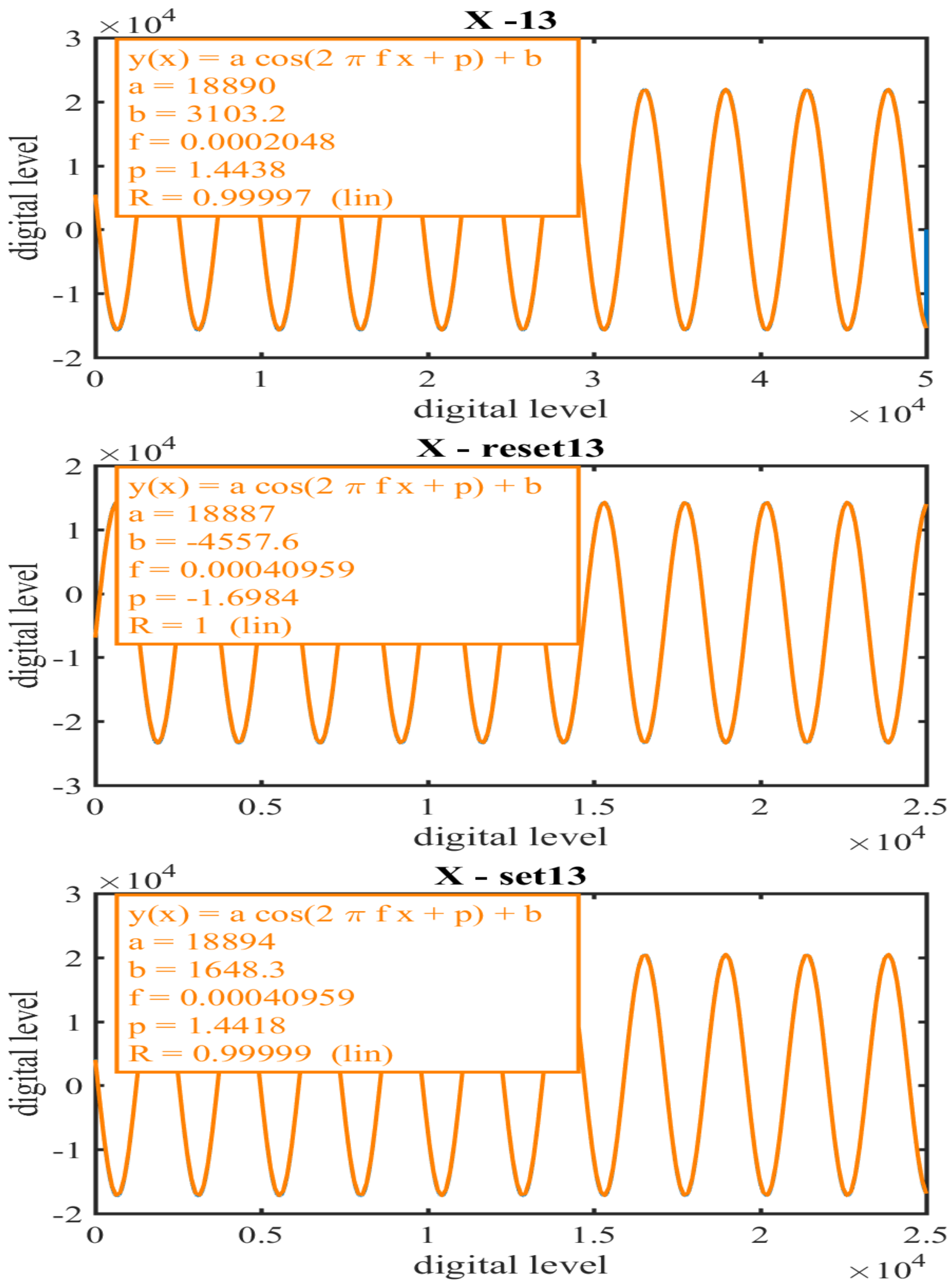


Figure 9.6: Amplitudes – Overall, Set, and Reset for data00013.txt

APPENDIX: F

OBJECTIVE 5 | TO DETERMINE THE NOISE |

F.1

TEST_PARTITION :

```
clc
close all
clear all
No_Of_Part=20;


for ii = 1:14
% ii=1;
Index=ii-1;
Label=sprintf('%05d',Index);
Data = load(['data' Label '.txt']);
Column_Start=1;
    for jj=1:32
        temp=Data(1,jj);
        if (temp~=0)
            Column_Start=jj;
            break
        end
    end
[r,c]=size(Data);

x=Data(:,Column_Start);
y=Data(:,Column_Start+1);
z=Data(:,Column_Start+2);

x=V2B(r,x);
y=V2B(r,y);
z=V2B(r,z);

figure
plot(x)
hold on;
plot(y)
plot(z)
[c,r]=size(x);
nr=floor(r/No_Of_Part);
X_Part=zeros(nr,No_Of_Part);
Y_Part=zeros(nr,No_Of_Part);
Z_Part=zeros(nr,No_Of_Part);
ff=1;
for jj=1:No_Of_Part
    for kk=1:nr
        X_Part(kk,jj)=x(ff);
        Y_Part(kk,jj)=y(ff);
        Z_Part(kk,jj)=z(ff);
        ff=ff+1;
    end
end
```

```
save(['Data_Part' Label],'X_Part','Y_Part','Z_Part');

end
```

Here V2B is function which converts a set of raw points into the corresponding values of amplitude. The code for V2B is given below.

```
function Vax = V2B (n, x)
 k=1;
 for ii = 1:(n-1)/2
    ii1=ii-1;
       Vax(k) = (x(2*ii+1)-x(2*ii))/2;
      k=k+1;
       Vax(k) = (x(2*ii1+1)-x(2*ii))/2;
      k=k+1;
 end
end
```

F.2

TEST NOISE:

```
clc
clear all
close all

defaultpaperfig
Avg_Noise_Density_X=zeros(1,14);
Avg_Noise_Density_Y=zeros(1,14);
Avg_Noise_Density_Z=zeros(1,14);
RMS_Noise_X=zeros(1,14);
RMS_Noise_Y=zeros(1,14);
RMS_Noise_Z=zeros(1,14);
STD_X=zeros(1,14);
STD_Y=zeros(1,14);
STD_Z=zeros(1,14);

for cc=1:14
  index=cc-1;
label=sprintf('%05d',index);
load (['Data_Part' label '.mat']);
[r,c]=size(X_Part);
PFT_x=zeros(r,c);
PFT_y=zeros(r,c);
PFT_z=zeros(r,c);
TS=0.002048;
FS=1/TS;
dF=FS/r;
F=dF*(0:((r/2)));
T=TS*(0:(r*c-1));
x_net=zeros(1,r*c);
y_net=zeros(1,r*c);
z_net=zeros(1,r*c);

kk=1;
for ii = 1:c
    for jj=1:r
```

```
        x_net(kk)=X_Part(jj,ii);
        y_net(kk)=Y_Part(jj,ii);
        z_net(kk)=Z_Part(jj,ii);
        kk=kk+1;
    end
end


figure
plot(T,x_net)
xlabel('time (sec)')
ylabel('digital level')
name=num2str(index);
title(strcat('X Amplitude - 999980 samples :',name))
name=['X raw data',name];
print(name,'-dpng')

figure
plot(T,y_net)
xlabel('time (sec)')
ylabel('digital level')
name=num2str(index);
title(strcat('Y Amplitude - 999980 samples :',name))
name=['Y raw data',name];
print(name,'-dpng')

figure
plot(T,z_net)
xlabel('time (sec)')
ylabel('digital level')
name=num2str(index);
title(strcat('Z Amplitude - 999980 samples :',name))
name=['Z raw data',name];
print(name,'-dpng')


for ii = 1:c
        mx  = mean(X_Part(:,ii));
        my  = mean(Y_Part(:,ii));
        mz  = mean(Z_Part(:,ii));
  PFT_x(:,ii)=abs(fft(X_Part(:,ii)-mx)).^2/(r^2);%this discards the DC
value,
  PFT_y(:,ii)=abs(fft(Y_Part(:,ii)-my)).^2/(r^2);%without which we would
need to
  PFT_z(:,ii)=abs(fft(Z_Part(:,ii)-mz)).^2/(r^2);%explicitly discard the
first point
end
PFT_mean_x=mean(PFT_x,2);
PFT_mean_x=PFT_mean_x(1:ceil(r/2));
PFT_mean_x(2:end)=2*PFT_mean_x(2:end);
PFT_mean_y=mean(PFT_y,2);
PFT_mean_y=PFT_mean_y(1:ceil(r/2));
PFT_mean_y(2:end)=2*PFT_mean_y(2:end);
PFT_mean_z=mean(PFT_z,2);
PFT_mean_z=PFT_mean_z(1:ceil(r/2));
PFT_mean_z(2:end)=2*PFT_mean_z(2:end);
% figure
% loglog(F,PFT_mean_x)
% title('X axis - Power vs frequency')
% xlabel('frequency (HZ)')
% ylabel('mean Power (signal^2)')
% figure
% loglog(F,PFT_mean_y)
% title('Y axis - Power vs frequency')
% xlabel('frequency (HZ)')
% ylabel('mean Power (signal^2)')
% figure
% loglog(F,PFT_mean_z)
% title('Z axis - Power vs frequency')
% xlabel('frequency (HZ)')
% ylabel('mean Power (signal^2)')

Noise_density_x=sqrt(PFT_mean_x/dF);
Noise_density_y=sqrt(PFT_mean_y/dF);
Noise_density_z=sqrt(PFT_mean_z/dF);
figure
loglog(F,Noise_density_x)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
name=num2str(index);
title(strcat('X axis - Noise density vs frequency :',name))
name=['X axis Noise density vs frequency',name];
print(name,'-dpng')

figure
loglog(F,Noise_density_y)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
name=num2str(index);
title(strcat('Y axis - Noise density vs frequency :',name))
name=['Y axis Noise density vs frequency',name];
print(name,'-dpng')

figure
loglog(F,Noise_density_z)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
name=num2str(index);
title(strcat('Z axis - Noise density vs frequency :',name))
name=['Z axis Noise density vs frequency',name];
print(name,'-dpng')

st=floor(0.5/dF);
ed=floor(37/dF);

jj=1;
avg_temp_x=zeros(ed-st,1);
avg_temp_y=zeros(ed-st,1);
avg_temp_z=zeros(ed-st,1);
for ii = st:ed
avg_temp_x(jj)=Noise_density_x(ii);
avg_temp_y(jj)=Noise_density_y(ii);
avg_temp_z(jj)=Noise_density_z(ii);
jj=jj+1;
end
Avg_Noise_Density_X(cc)=mean(avg_temp_x);
Avg_Noise_Density_Y(cc)=mean(avg_temp_y);
Avg_Noise_Density_Z(cc)=mean(avg_temp_z);

%Noise_density_x(1:end,1)=Noise_density_x(1:end,1)

STD_X(cc)=std(x_net);
```

```
STD_Y(cc)=std(y_net);
STD_Z(cc)=std(z_net);
RMS_Noise_X(cc)=sqrt(trapz(F,Noise_density_x.^2));
RMS_Noise_Y(cc)=sqrt(trapz(F,Noise_density_y.^2));
RMS_Noise_Z(cc)=sqrt(trapz(F,Noise_density_z.^2));
End
save('Noise2','RMS_Noise_X','RMS_Noise_Y','RMS_Noise_Z','STD_X','STD_Y','ST
D_Z','Avg_Noise_Density_X','Avg_Noise_Density_Y','Avg_Noise_Density_Z');
```

F.3

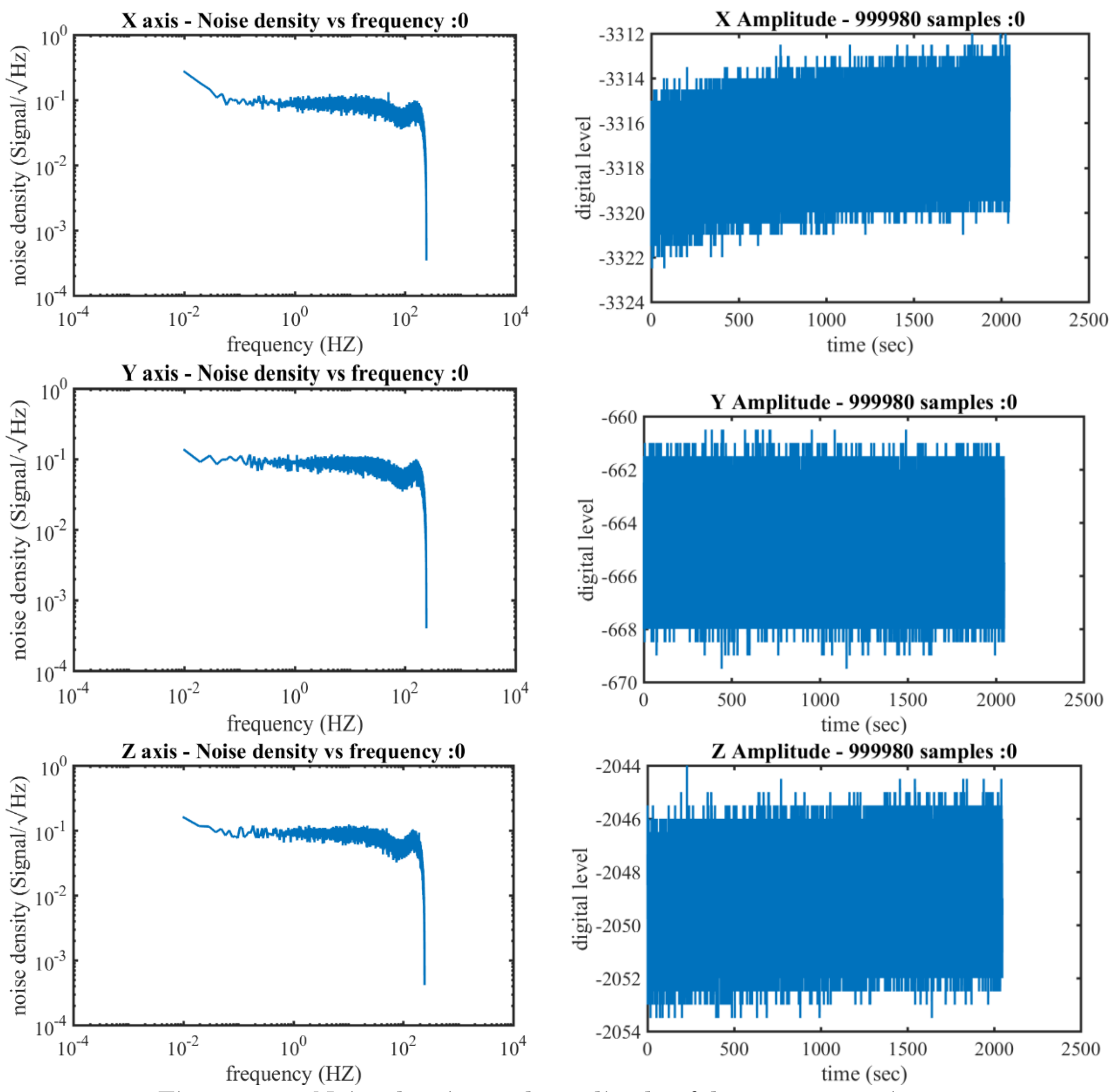


Figure 10.3: Noise density and amplitude of data00000.txt, i.e sensor s17

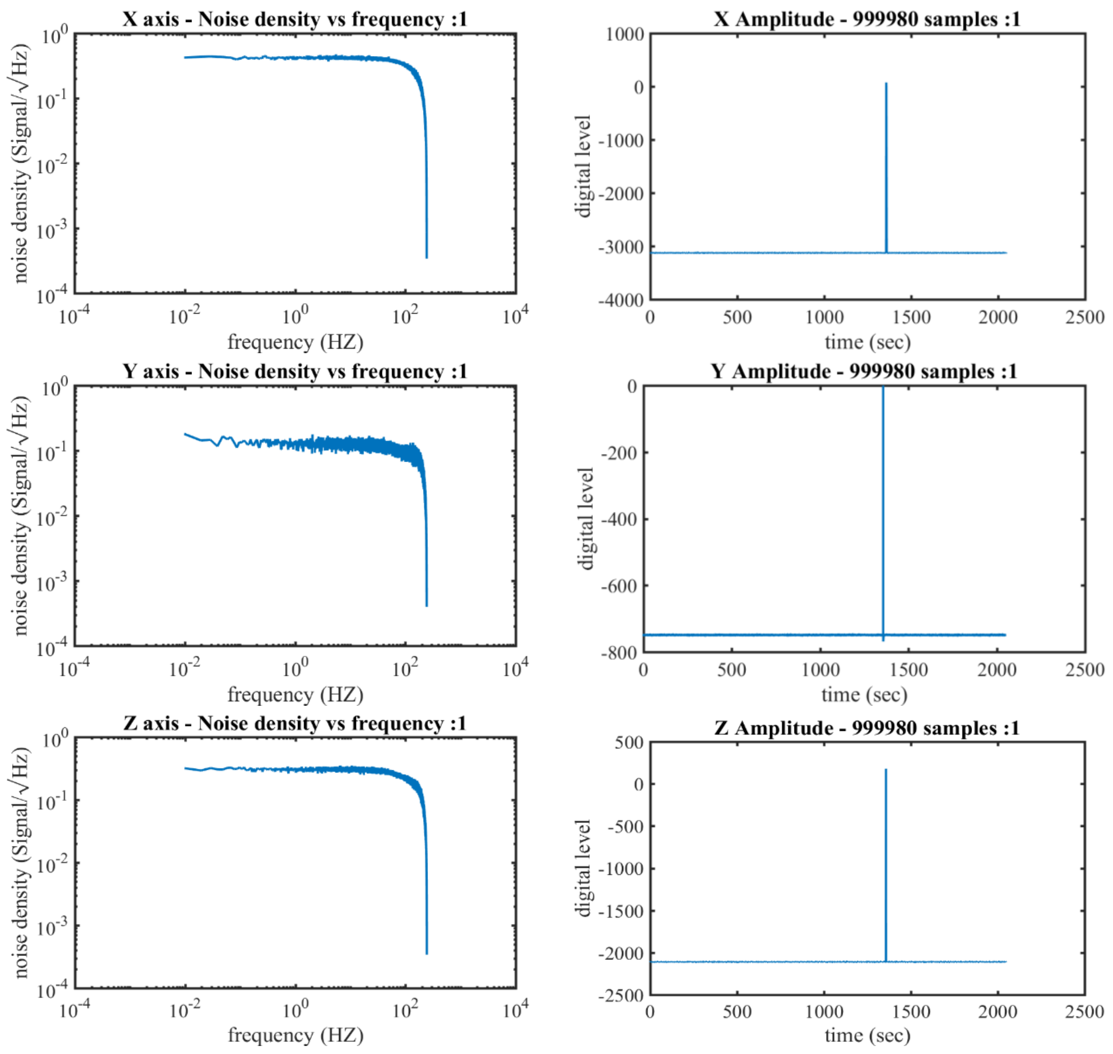
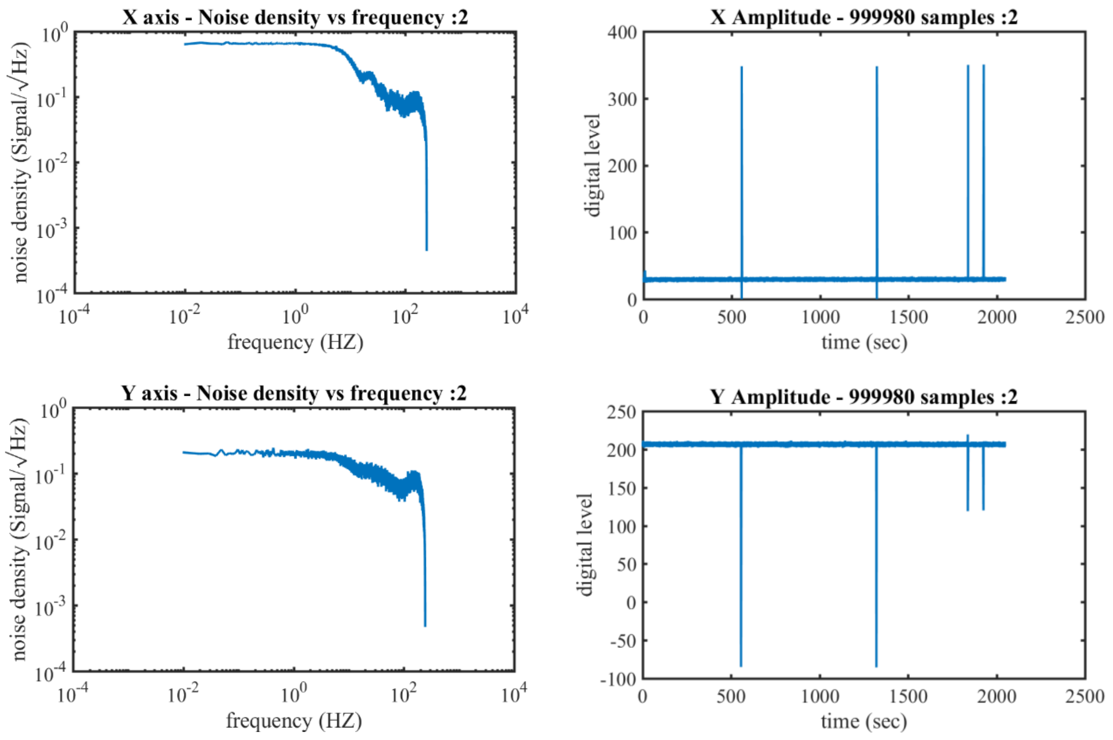


Figure 10.4: Noise density and amplitude of data00001.txt, i.e sensor s7



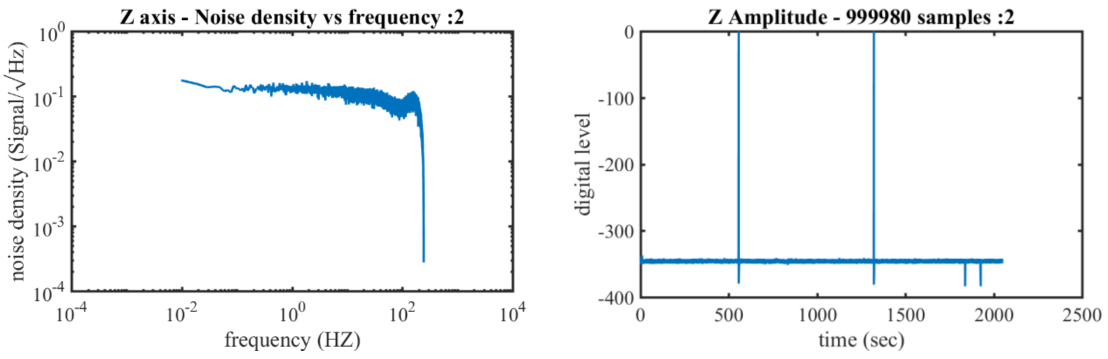


Figure 10.5: Noise density and amplitude of data00002.txt, i.e sensor s5

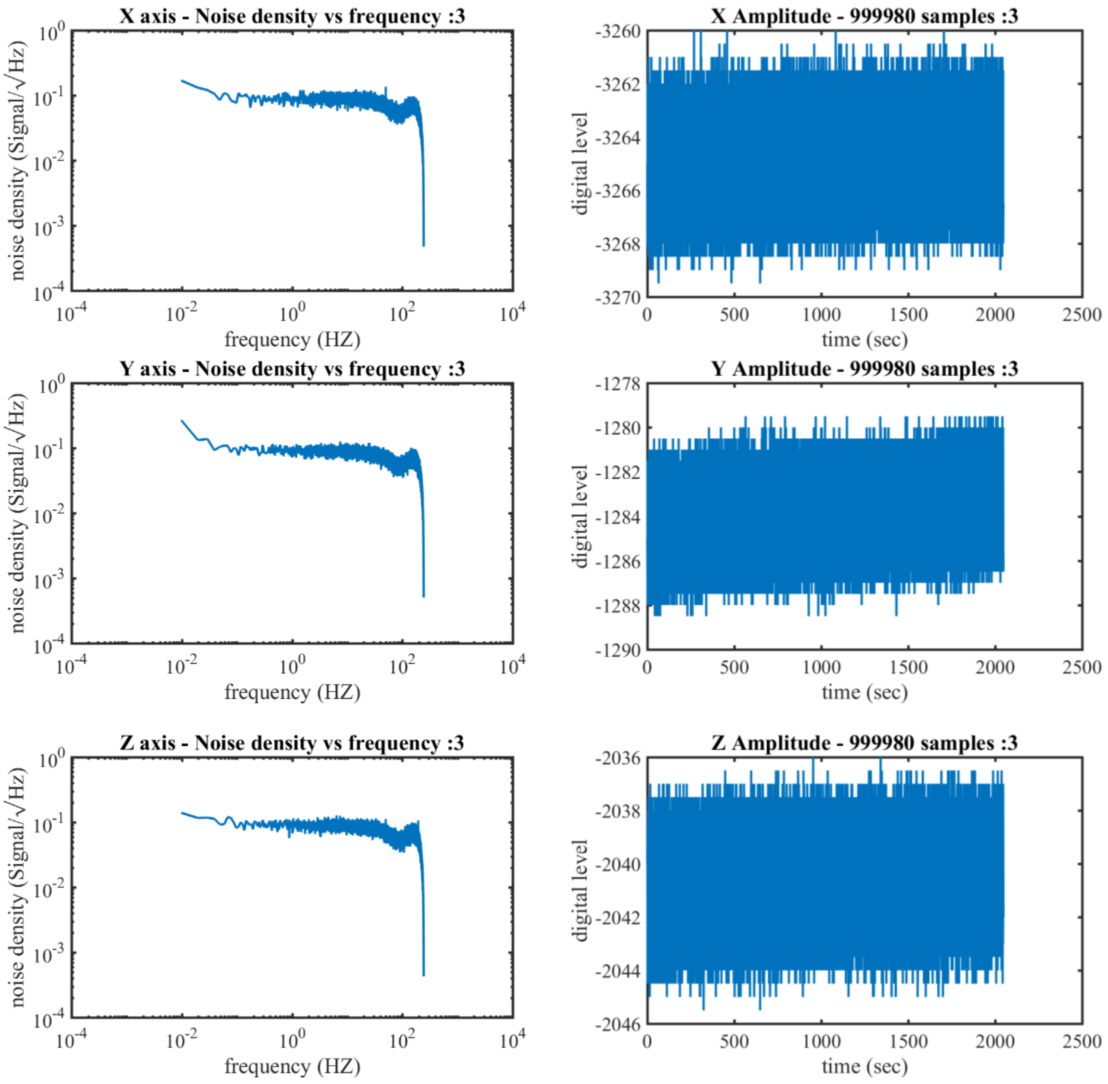


Figure 10.6: Noise density and amplitude of data00003.txt, i.e sensor s14

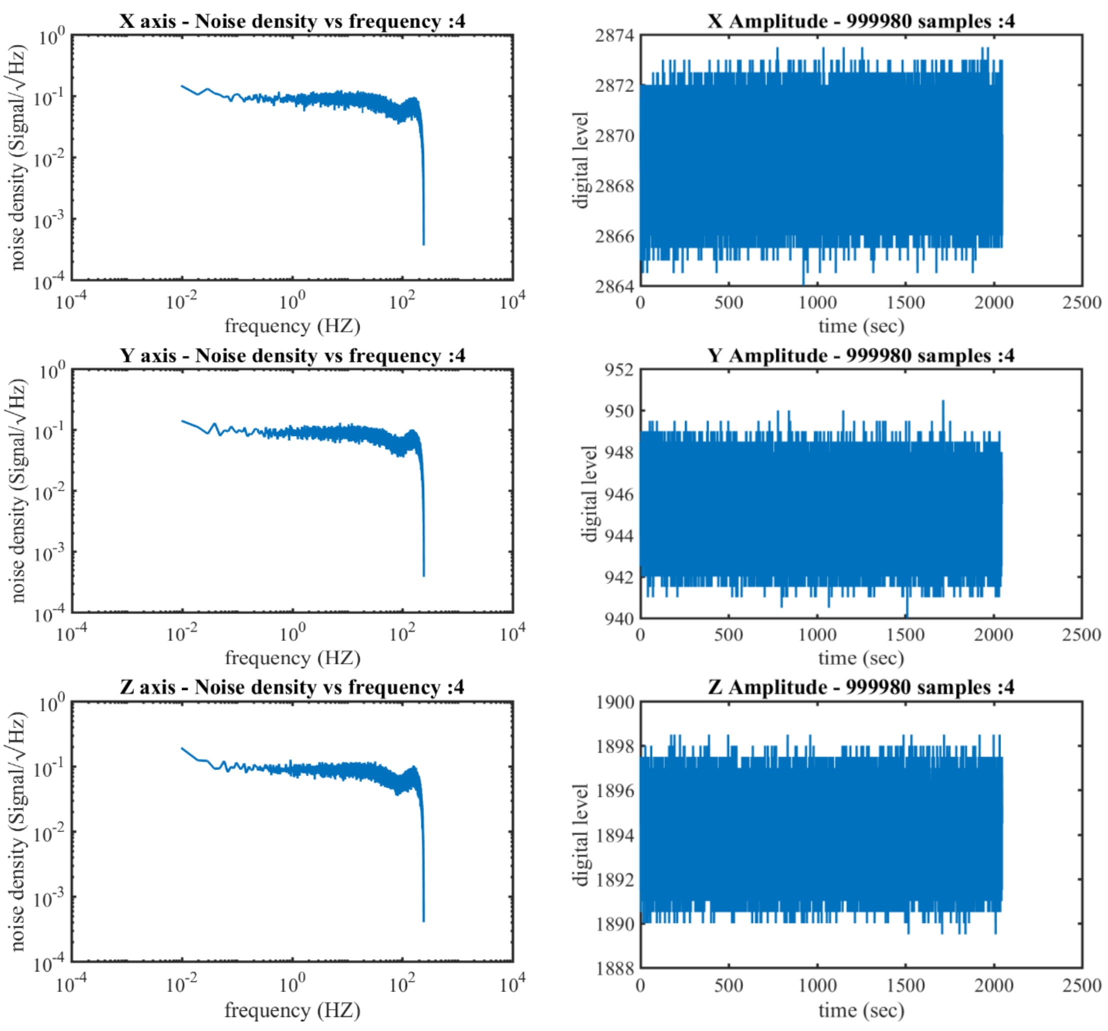
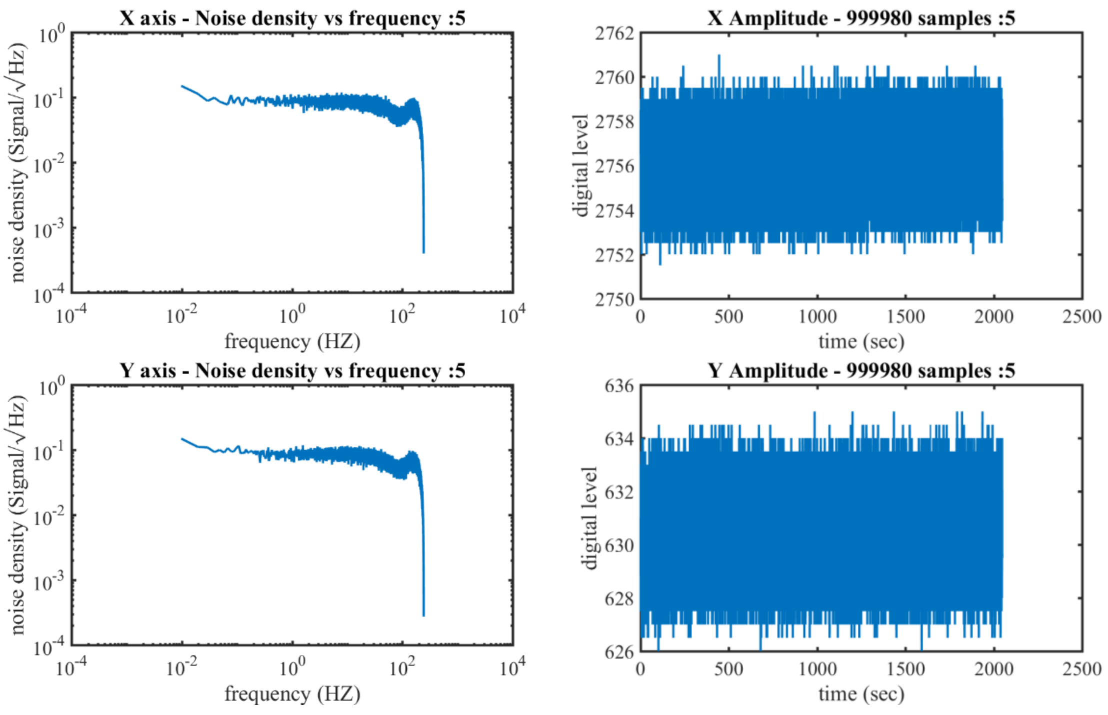


Figure 10.7: Noise density and amplitude of data00004.txt, i.e sensor s10



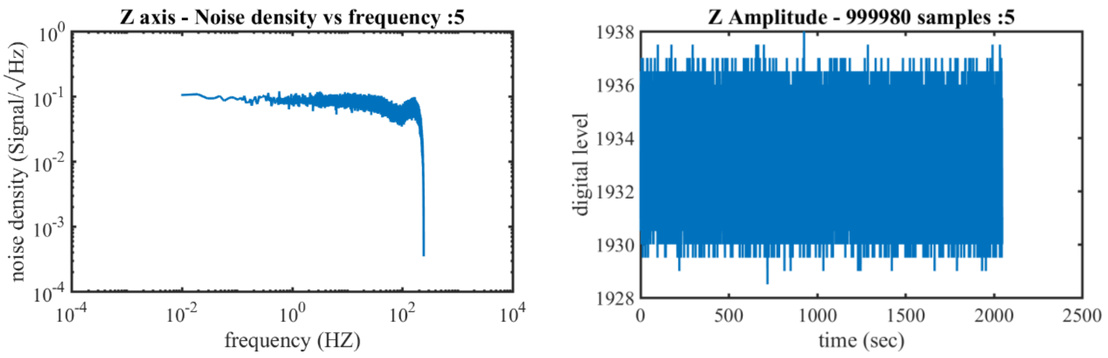


Figure 10.8: Noise density and amplitude of data00005.txt, i.e sensor s6

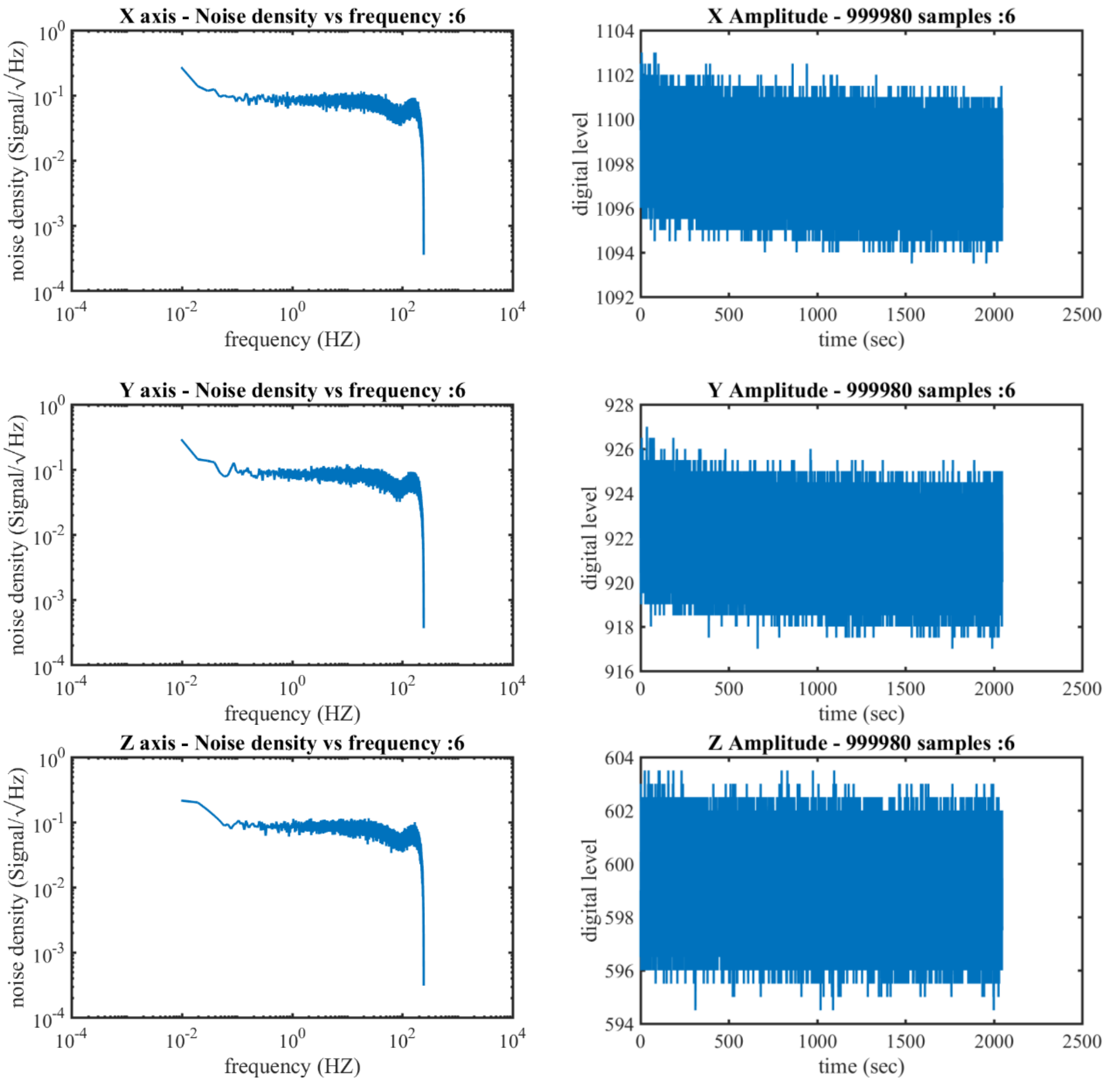


Figure 10.9: Noise density and amplitude of data00006.txt, i.e sensor s8

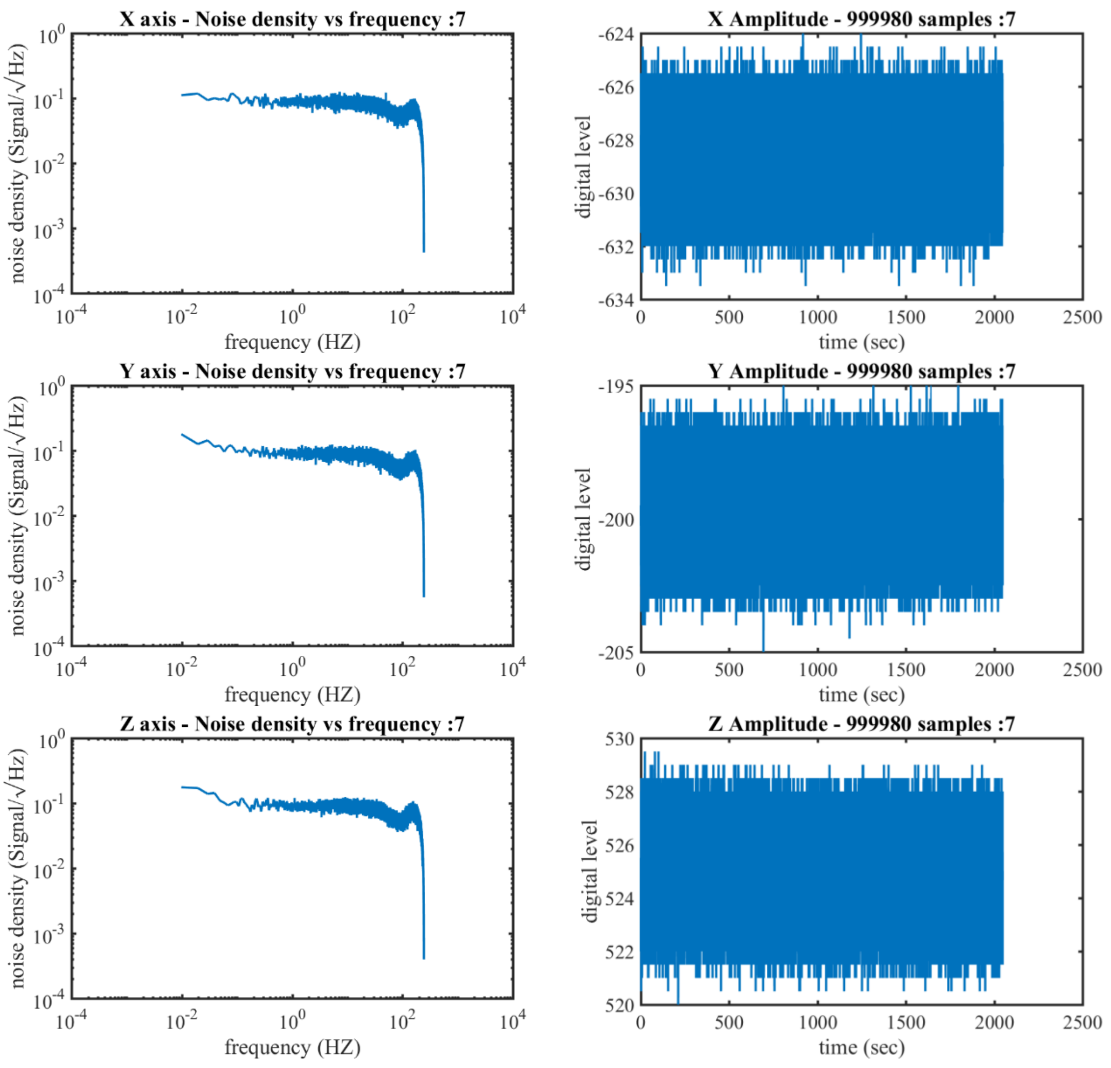
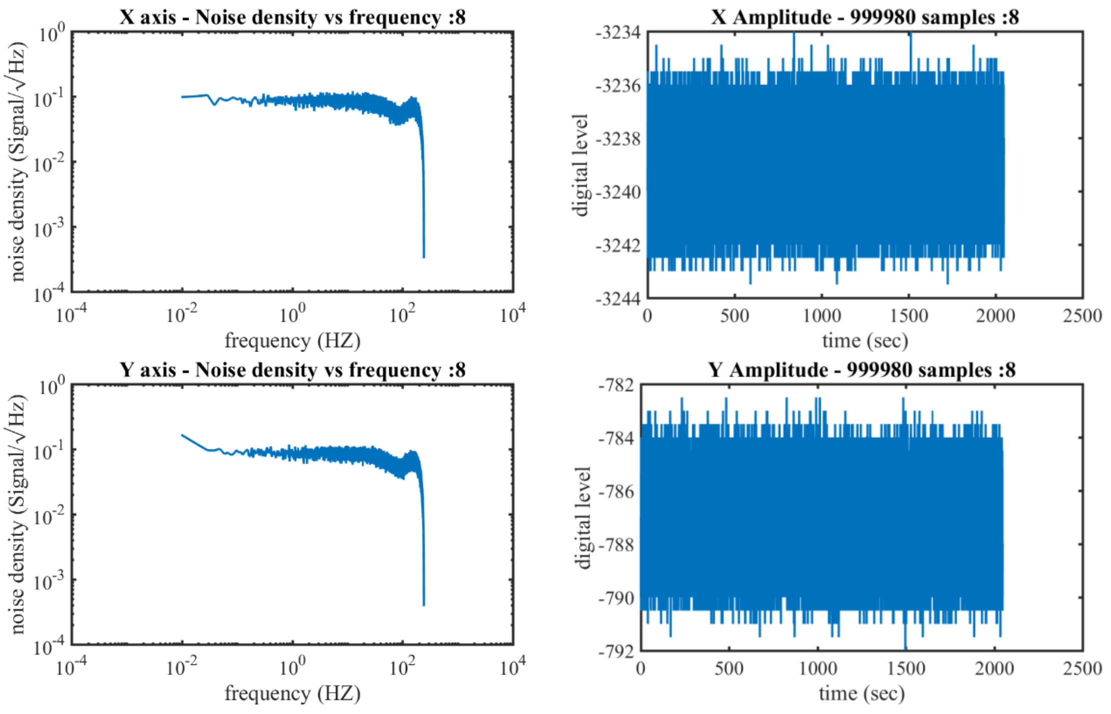


Figure 11.0: Noise density and amplitude of data00007.txt, i.e sensor s1



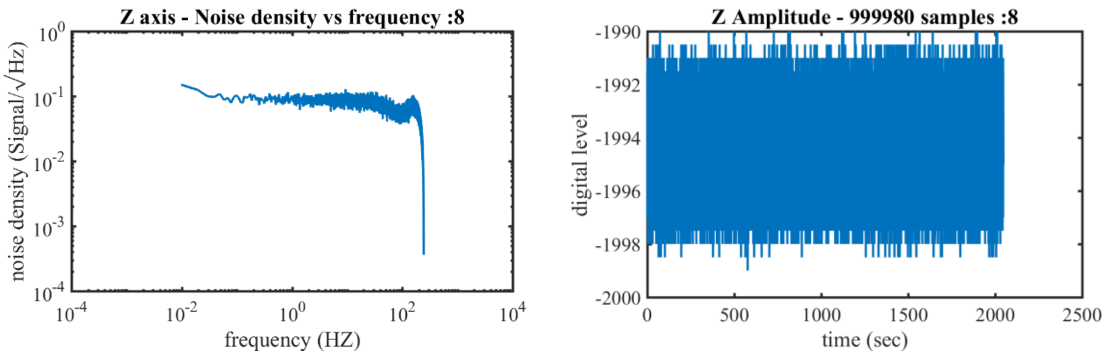


Figure 11.1: Noise density and amplitude of data00008.txt, i.e sensor s16

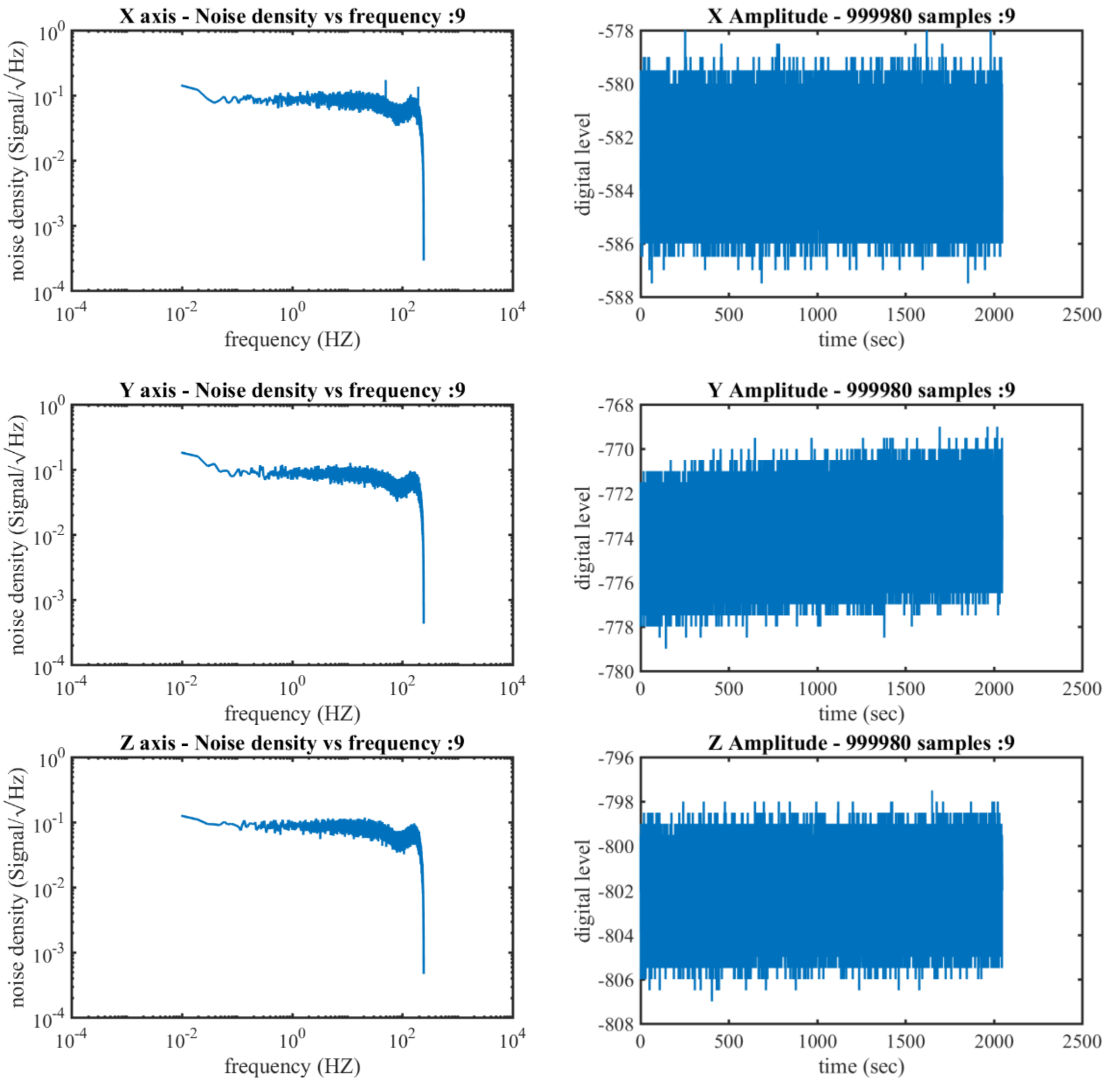


Figure 11.2: Noise density and amplitude of data00009.txt, i.e sensor s2

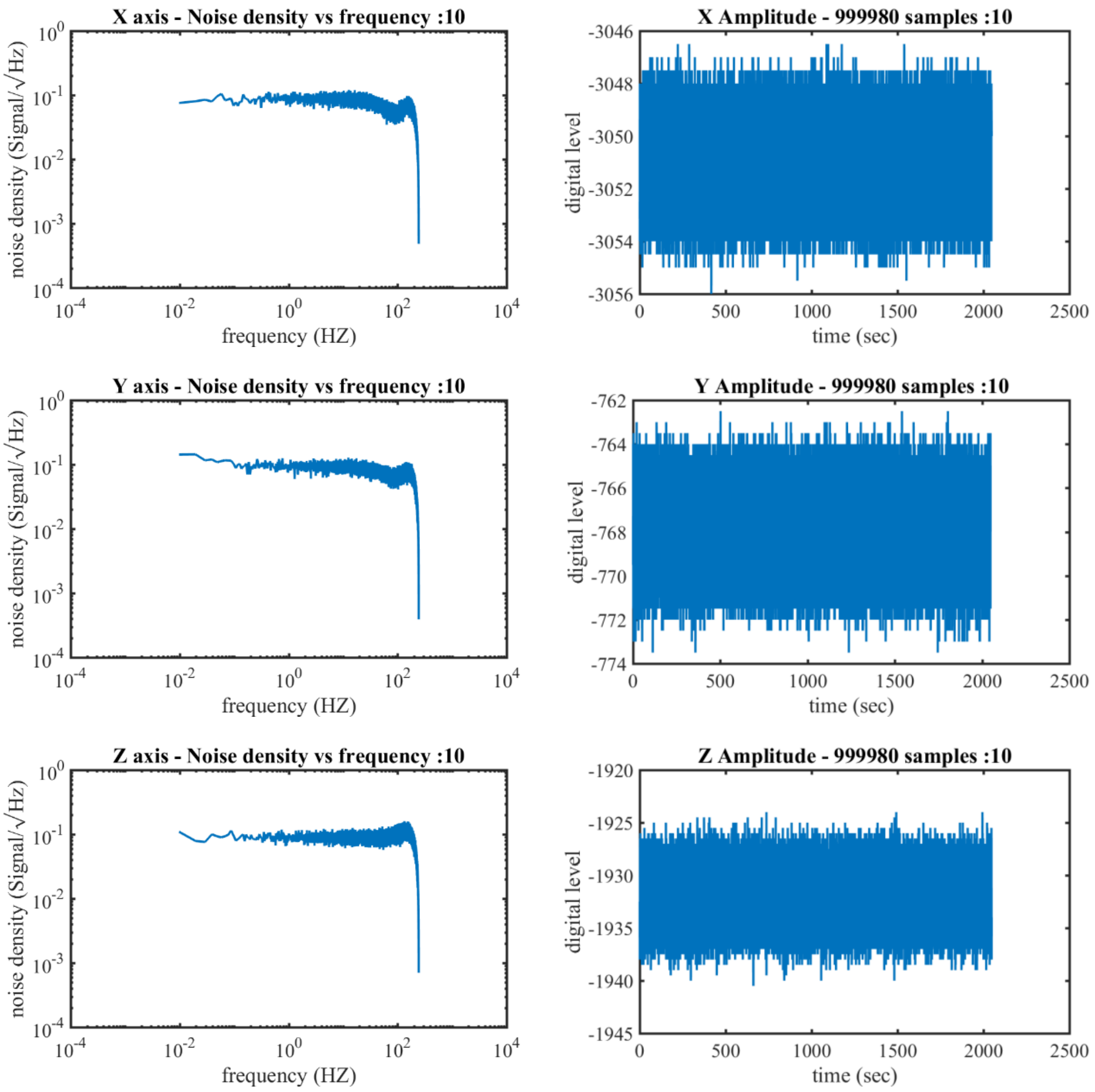
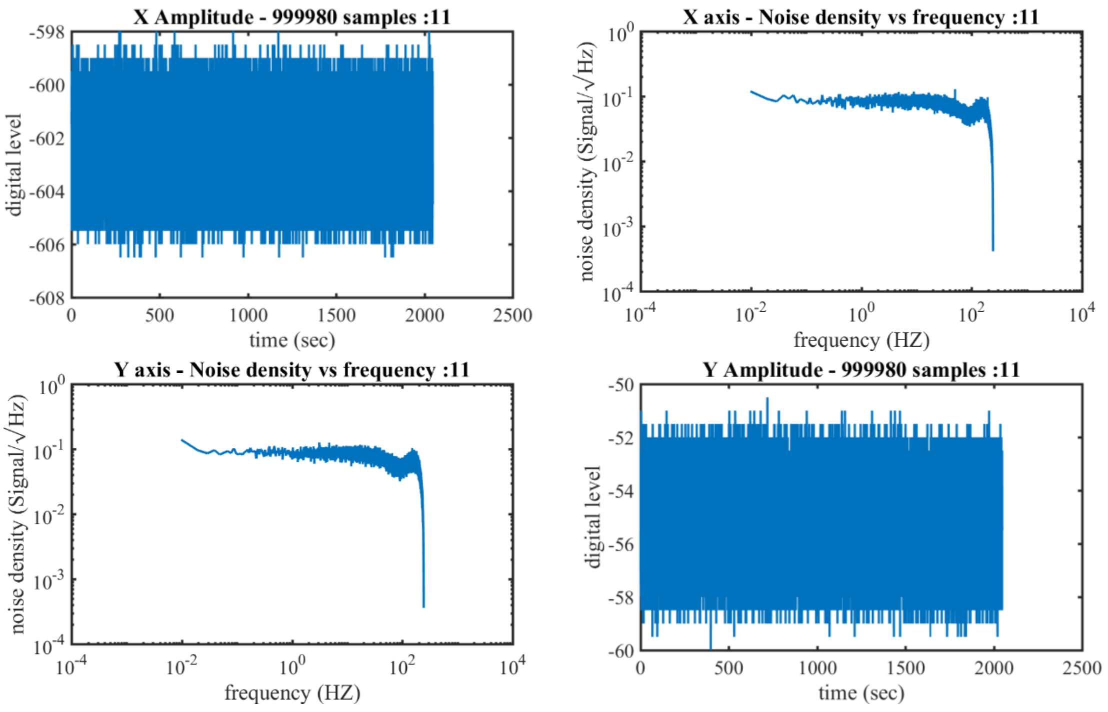


Figure 11.3: Noise density and amplitude of data00010.txt, i.e sensor s12



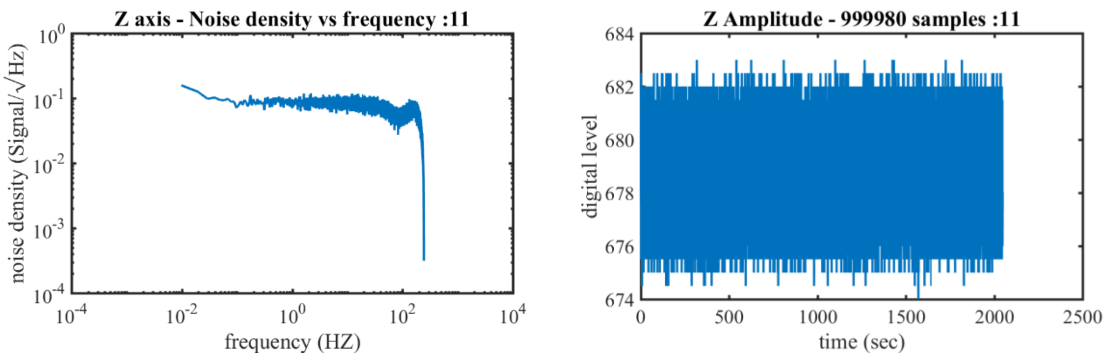


Figure 11.4: Noise density and amplitude of data00011.txt, i.e sensor s3

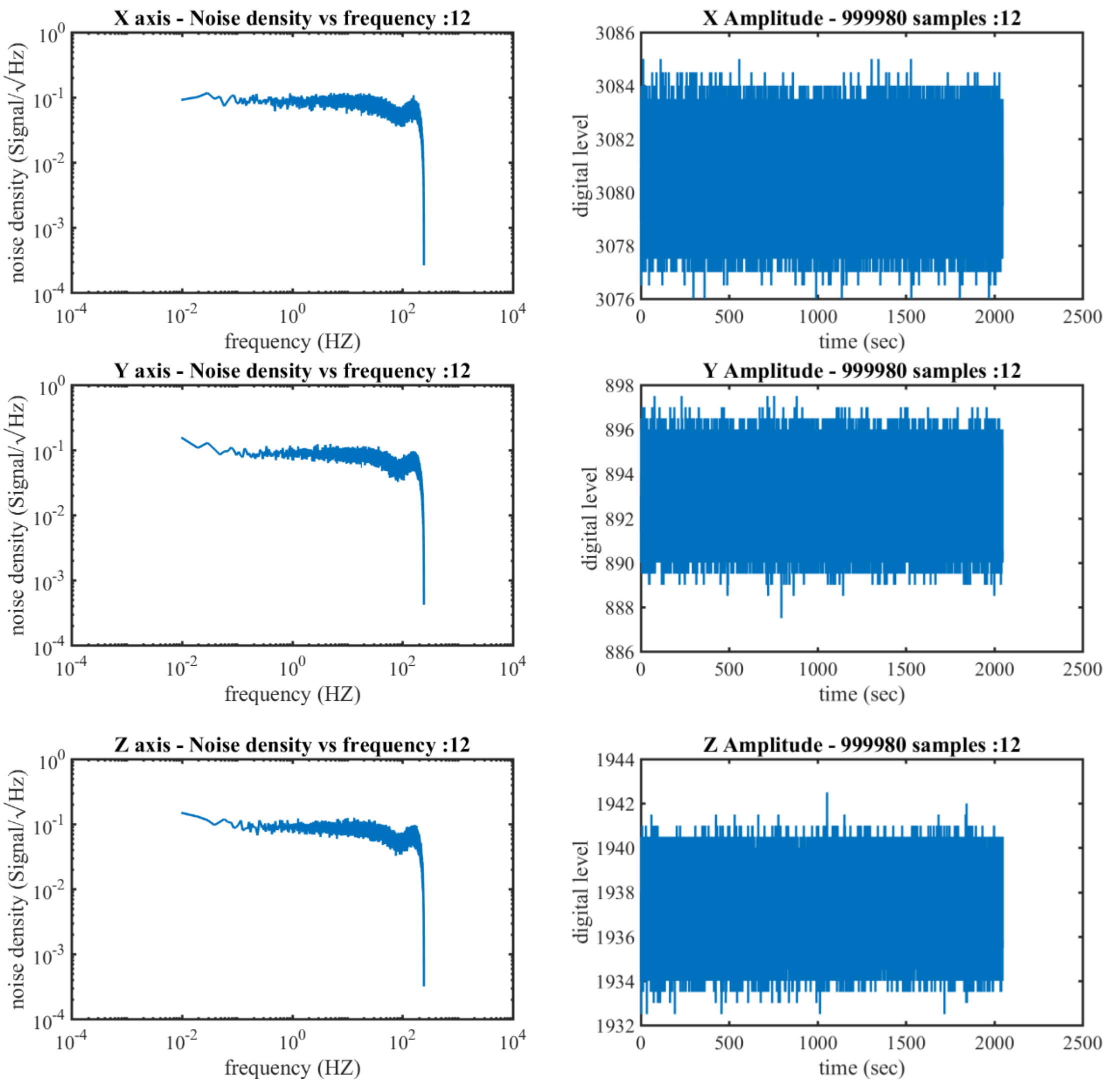


Figure 11.5: Noise density and amplitude of data00012.txt, i.e sensor s15

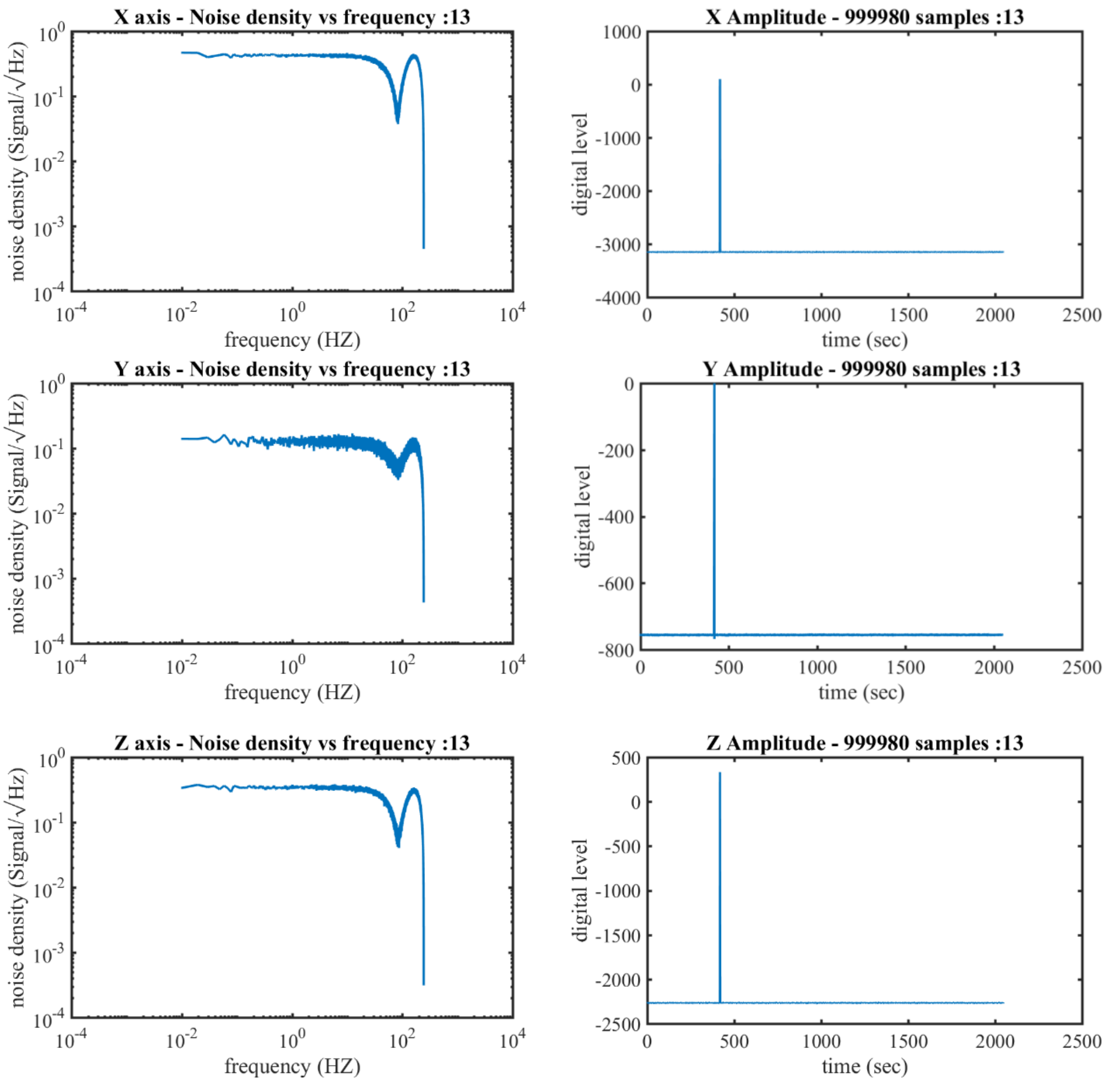


Figure 11.6: Noise density and amplitude of data00013.txt, i.e sensor s13

F.4

THRESHOLD NOISE:

```
close all
clear all
clc
load('Noise2.mat');

for jj=1:14

      X_Noise_threshold_temp(1,jj)=Avg_Noise_Density_X(1,jj)*35.97;
      Y_Noise_threshold_temp(1,jj)=Avg_Noise_Density_Y(1,jj)*35.97;
      Z_Noise_threshold_temp(1,jj)=Avg_Noise_Density_Z(1,jj)*35.97;

      X_RMS_temp(1,jj)=RMS_Noise_X(1,jj)*35.97;
      Y_RMS_temp(1,jj)=RMS_Noise_Y(1,jj)*35.97;
      Z_RMS_temp(1,jj)=RMS_Noise_Z(1,jj)*35.97;


end
```

```
jj=1;
for ii = 1:14
    if(ii~= 3 && ii~= 8 && ii~=11 && ii~=14 && ii~=2 && ii~=12)
       X_Noise_threshold(1,jj)=X_Noise_threshold_temp(1,ii);
       Y_Noise_threshold(1,jj)=Y_Noise_threshold_temp(1,ii);
       Z_Noise_threshold(1,jj)=Z_Noise_threshold_temp(1,ii);

       X_RMS(1,jj)=X_RMS_temp(1,ii);
       Y_RMS(1,jj)=Y_RMS_temp(1,ii);
       Z_RMS(1,jj)=Z_RMS_temp(1,ii);

       jj=jj+1;
    end
end

figure
plot(X_Noise_threshold)
hold on
plot(Y_Noise_threshold)
hold on
plot(Z_Noise_threshold)
xlabel('Sensors')
ylabel('Micro gauss/\surdHz')
legend('X spectral noise density','Y spectral noise density','Z spectral
noise density')
title('X, Y and Z Spectral Noise Density')
print('Spectral Noise Density','-dpng')

figure
plot(X_RMS)
hold on
plot(Y_RMS)
hold on
plot(Z_RMS)
xlabel('Sensors')
ylabel('Micro gauss')
legend('X RMS Noise','Y RMS Noise','Z RMS Noise')
title('X, Y and Z RMS Noise')
print('RMS Noise','-dpng')
```

F.5

PLOT DATA:

```
clc
close all;
clear all;
defaultpaperfig

d = load('data00000.txt');

[n,ccc]=size(d);
% dt = 0.002048;
% t = (0:(n-1))*dt;


Column_Start=1;
    for jj=1:32
        temp=d(1,jj);
        if (temp~=0)
```

```
            Column_Start=jj;
            break
        end
    end
x=d(:,Column_Start);
y=d(:,Column_Start+1);
z=d(:,Column_Start+2);
c=d(:,Column_Start+3);

offsetx = (x(2:end) + x(1:end-1)) /2;
offsety = (y(2:end) + y(1:end-1)) /2;
offsetz = (z(2:end) + z(1:end-1)) /2;

 k=1;
 for ii = 1:(n-1)/2
    ii1=ii-1;
       Vampx(k) = (x(2*ii+1)-x(2*ii))/2;
       Vampy(k) = (y(2*ii+1)-y(2*ii))/2;
       Vampz(k) = (z(2*ii+1)-z(2*ii))/2;
      k=k+1;
       Vampx(k) = (x(2*ii1+1)-x(2*ii))/2;
       Vampy(k) = (y(2*ii1+1)-y(2*ii))/2;
       Vampz(k) = (z(2*ii1+1)-z(2*ii))/2;
      k=k+1;
 end

% figure
% t=1:1000000;
% plot(t,c,'-')
% title ('counter')

figure
plot(x)
hold on;
plot(offsetx,'r')
hold on;
plot(Vampx)
title('Raw data - X axis ')
xlabel('Digital level')
ylabel('Digital level')
legend('Both Set-Reset','Bridge Offset','Amplitude')
print('Raw data X axis','-dpng')

figure
plot(y)
hold on;
plot(offsety,'r')
hold on;
plot(Vampy)
title('Raw data - Y axis ')
xlabel('Digital level')
ylabel('Digital level')
legend('Both Set-Reset','Bridge Offset','Amplitude')
print('Raw data Y axis','-dpng')


figure
plot(z)
hold on;
plot(offsetz,'r')
hold on;
```

```
plot(Vampz)
title('Raw data - Z axis ')
xlabel('Digital level')
ylabel('Digital level')
legend('Both Set-Reset','Bridge Offset','Amplitude')
print('Raw data Z axis','-dpng')
```

F.6

TEST_NOISE:

```
clc
clear all
close all

defaultpaperfig

d=load('data00000.txt');
d_zgc=load('data00001.txt');
Column_Start=1;
    for jj=1:32
        temp=d(1,jj);
        if (temp~=0)
            Column_Start=jj;
            break
        end
    end
x=d(:,Column_Start);
y=d(:,Column_Start+1);
z=d(:,Column_Start+2);

[r,c]=size(x);

x=V2B(r,x);
y=V2B(r,y);
z=V2B(r,z);



x_zgc=d_zgc(:,Column_Start);
y_zgc=d_zgc(:,Column_Start+1);
z_zgc=d_zgc(:,Column_Start+2);

x_zgc=V2B(r,x_zgc);
y_zgc=V2B(r,y_zgc);
z_zgc=V2B(r,z_zgc);

x_mg=x-x_zgc;
y_mg=y-y_zgc;
z_mg=z-z_zgc;

TS=0.002048;
FS=1/TS;
dF=FS/r;
F=dF*(0:((r-1)/2));
T=TS*(0:(r-1));

% outside ZGC
```

```
    mx   = mean(x);
    my   = mean(y);
    mz   = mean(z);


  PFT_x=abs(fft(x-mx)).^2/(r^2);%this discards the DC value,
  PFT_y=abs(fft(y-my)).^2/(r^2);%without which we would need to
  PFT_z=abs(fft(z-mz)).^2/(r^2);%explicitly discard the first point

PFT_x=PFT_x(1:ceil(r/2));
PFT_x(2:end)=2*PFT_x(2:end);

PFT_y=PFT_y(1:ceil(r/2));
PFT_y(2:end)=2*PFT_y(2:end);

PFT_z=PFT_z(1:ceil(r/2));
PFT_z(2:end)=2*PFT_z(2:end);

%inside ZGC


  mx_zgc   = mean(x_zgc);
  my_zgc   = mean(y_zgc);
  mz_zgc   = mean(z_zgc);


  PFT_x_zgc=abs(fft(x_zgc-mx_zgc)).^2/(r^2);%this discards the DC value,
  PFT_y_zgc=abs(fft(y_zgc-my_zgc)).^2/(r^2);%without which we would need to
  PFT_z_zgc=abs(fft(z_zgc-mz_zgc)).^2/(r^2);%explicitly discard the first
point


PFT_x_zgc=PFT_x_zgc(1:ceil(r/2));
PFT_x_zgc(2:end)=2*PFT_x_zgc(2:end);

PFT_y_zgc=PFT_y_zgc(1:ceil(r/2));
PFT_y_zgc(2:end)=2*PFT_y_zgc(2:end);

PFT_z_zgc=PFT_z_zgc(1:ceil(r/2));
PFT_z_zgc(2:end)=2*PFT_z_zgc(2:end);

%Magnetic


  mx_mg   = mean(x_mg);
  my_mg   = mean(y_mg);
  mz_mg   = mean(z_mg);


  PFT_x_mg=abs(fft(x_mg-mx_mg)).^2/(r^2);%this discards the DC value,
  PFT_y_mg=abs(fft(y_mg-my_mg)).^2/(r^2);%without which we would need to
  PFT_z_mg=abs(fft(z_mg-mz_mg)).^2/(r^2);%explicitly discard the first
point


PFT_x_mg=PFT_x_mg(1:ceil(r/2));
PFT_x_mg(2:end)=2*PFT_x_mg(2:end);
```

```
PFT_y_mg=PFT_y_mg(1:ceil(r/2));
PFT_y_mg(2:end)=2*PFT_y_mg(2:end);

PFT_z_mg=PFT_z_mg(1:ceil(r/2));
PFT_z_mg(2:end)=2*PFT_z_mg(2:end);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Noise_density_x=sqrt(PFT_x/dF);
Noise_density_y=sqrt(PFT_y/dF);
Noise_density_z=sqrt(PFT_z/dF);

Noise_density_x_zgc=sqrt(PFT_x_zgc/dF);
Noise_density_y_zgc=sqrt(PFT_y_zgc/dF);
Noise_density_z_zgc=sqrt(PFT_z_zgc/dF);

Noise_density_x_mg=sqrt(PFT_x_mg/dF);
Noise_density_y_mg=sqrt(PFT_y_mg/dF);
Noise_density_z_mg=sqrt(PFT_z_mg/dF);

% Noise_density_x_mg=Noise_density_x-Noise_density_x_zgc;
% Noise_density_y_mg=Noise_density_y-Noise_density_y_zgc;
% Noise_density_z_mg=Noise_density_z-Noise_density_z_zgc;
figure
loglog(F,Noise_density_x)
hold on
loglog(F,Noise_density_x_zgc)
hold on
loglog(F,Noise_density_x_mg)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
title('X axis - Noise density vs frequency')
legend('Noise outside ZGC','Noise inside ZGC','Magnetic noise')
print('X axis Noise density vs frequency','-dpng')

figure
loglog(F,Noise_density_y)
hold on
loglog(F,Noise_density_y_zgc)
hold on
loglog(F,Noise_density_y_mg)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
title('Y axis - Noise density vs frequency')
legend('Noise outside ZGC','Noise inside ZGC','Magnetic noise')
print('Y axis Noise density vs frequency','-dpng')

figure
loglog(F,Noise_density_z)
hold on
loglog(F,Noise_density_z_zgc)
hold on
loglog(F,Noise_density_z_mg)
xlabel('frequency (HZ)')
ylabel('noise density (Signal/\surd{Hz})')
title('Z axis - Noise density vs frequency')
legend('Noise outside ZGC','Noise inside ZGC','Magnetic noise')
print('Z axis Noise density vs frequency','-dpng')
```

```
RMS_Noise_X=sqrt(trapz(F,Noise_density_x.^2));
RMS_Noise_Y=sqrt(trapz(F,Noise_density_y.^2));
RMS_Noise_Z=sqrt(trapz(F,Noise_density_z.^2));


RMS_Noise_X_zgc=sqrt(trapz(F,Noise_density_x_zgc.^2));
RMS_Noise_Y_zgc=sqrt(trapz(F,Noise_density_y_zgc.^2));
RMS_Noise_Z_zgc=sqrt(trapz(F,Noise_density_z_zgc.^2));


RMS_Noise_X_mg=sqrt(trapz(F,Noise_density_x_mg.^2));
RMS_Noise_Y_mg=sqrt(trapz(F,Noise_density_y_mg.^2));
RMS_Noise_Z_mg=sqrt(trapz(F,Noise_density_z_mg.^2));

%save('Noise2','RMS_Noise_X','RMS_Noise_Y','RMS_Noise_Z','STD_X','STD_Y','S
TD_Z','Avg_Noise_Density_X','Avg_Noise_Density_Y','Avg_Noise_Density_Z');
```

- _x86);

APPENDIX: G

OBJECTIVE 6 | THE ROBUSTNESS |

G.1

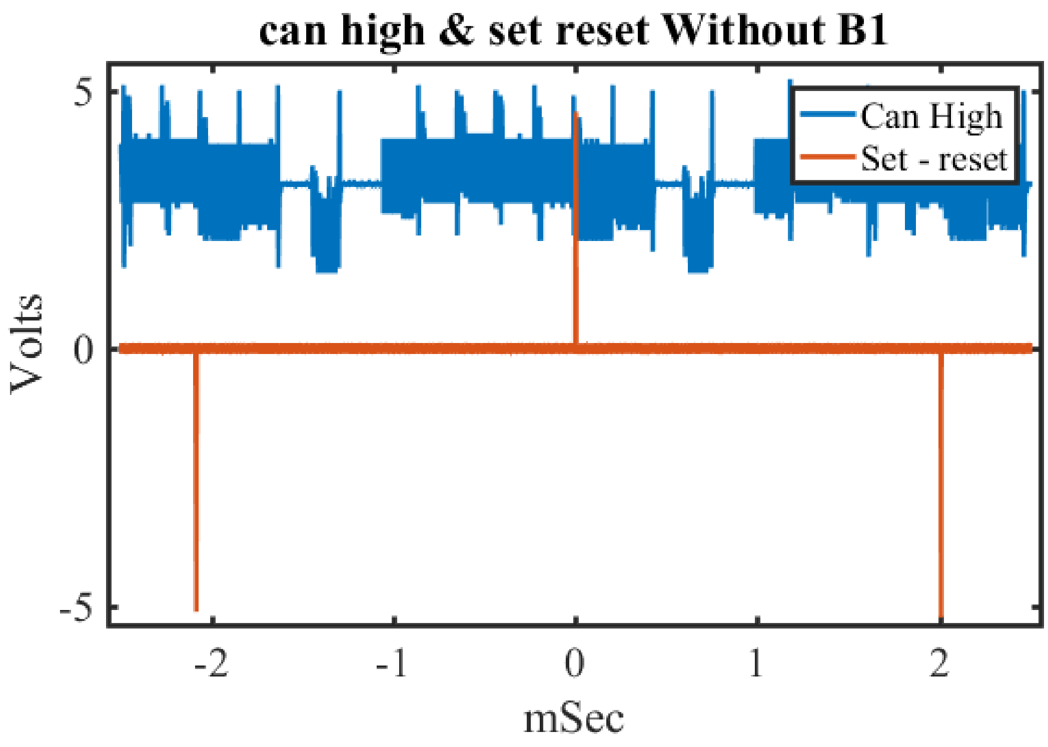


Figure 12.3 : Without B sensors 1

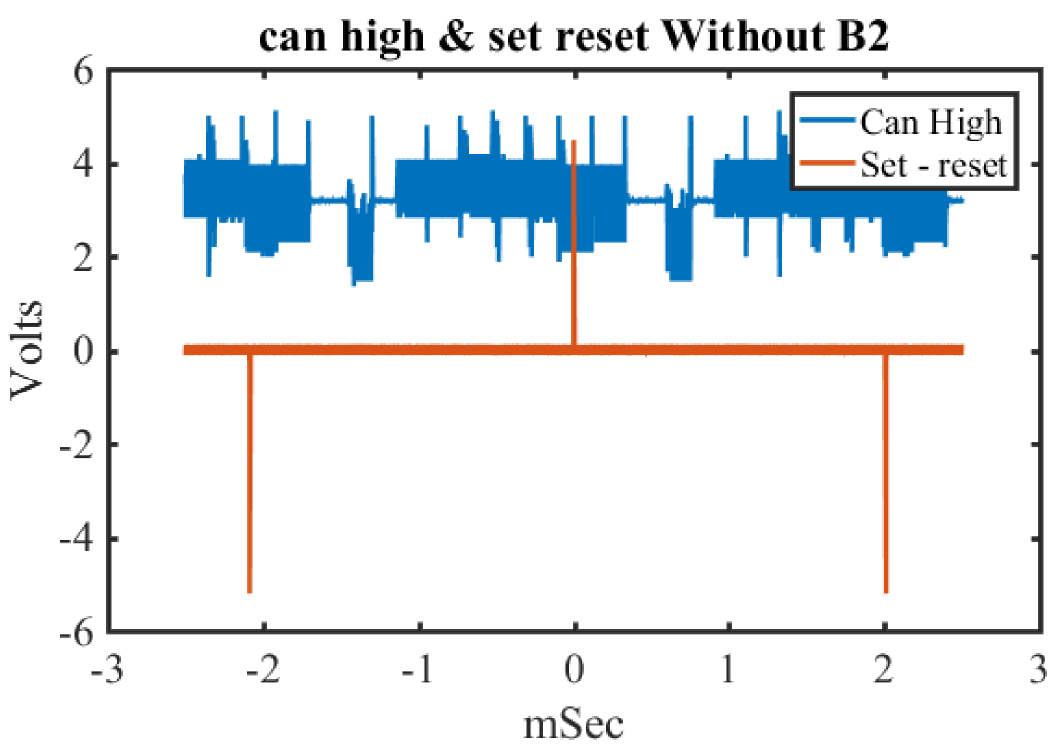


Figure 12.4 : Without B sensors 2

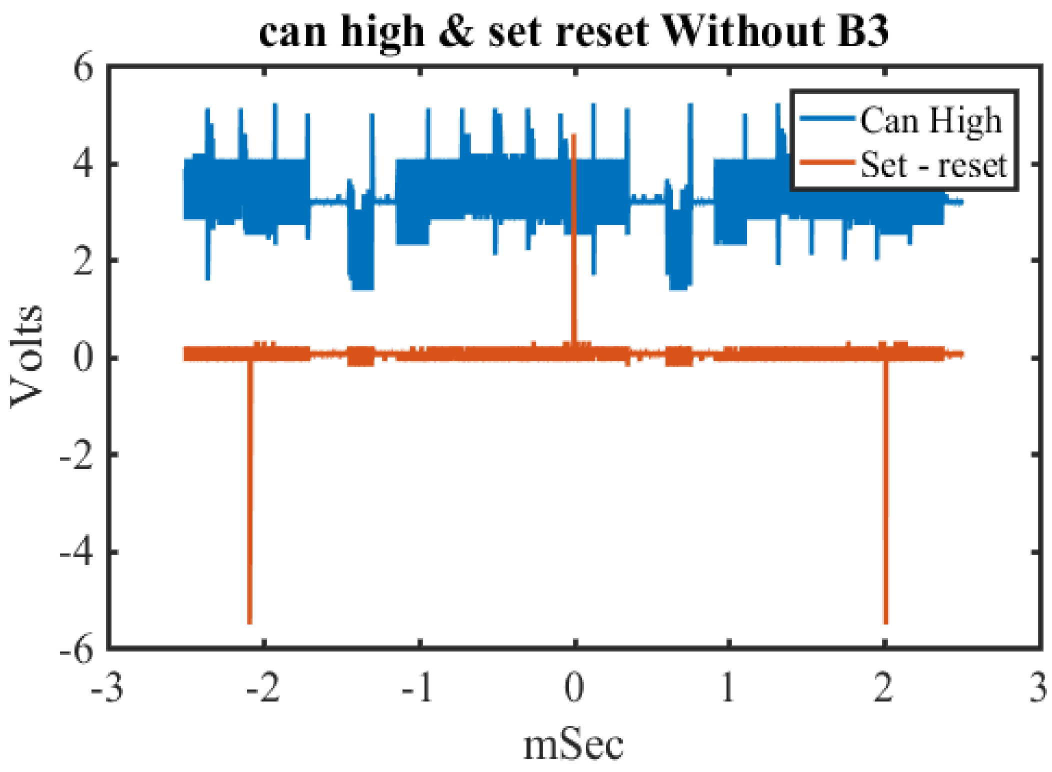


Figure 12.5 : Without B sensors 3

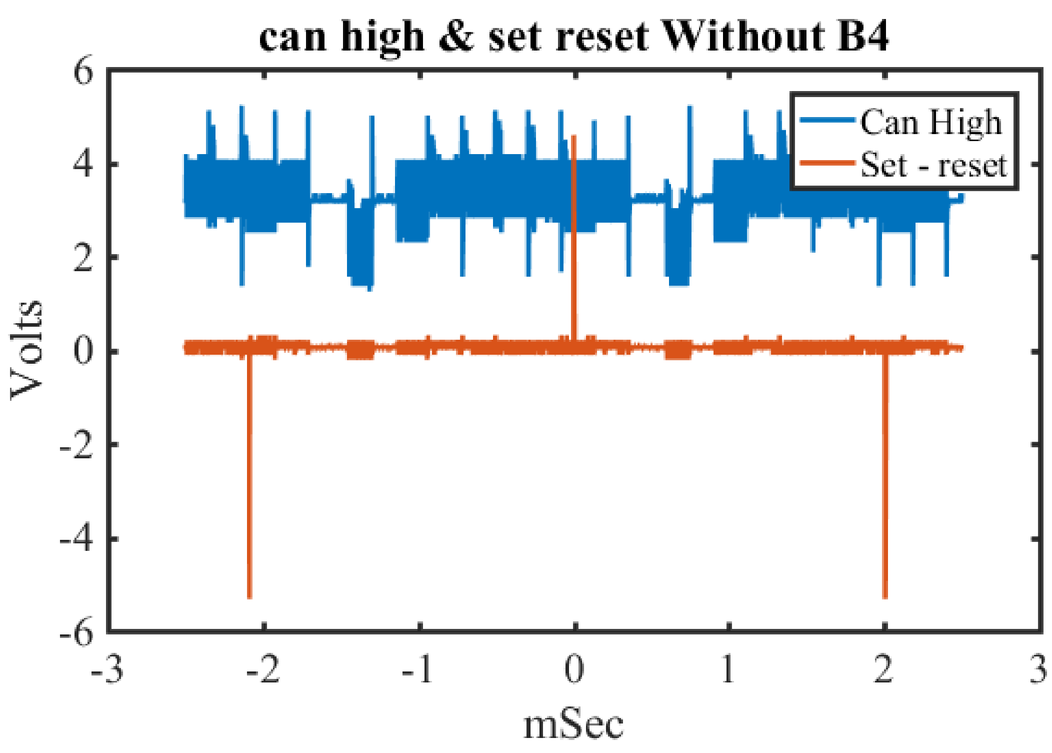


Figure 12.6 : Without B sensors 4

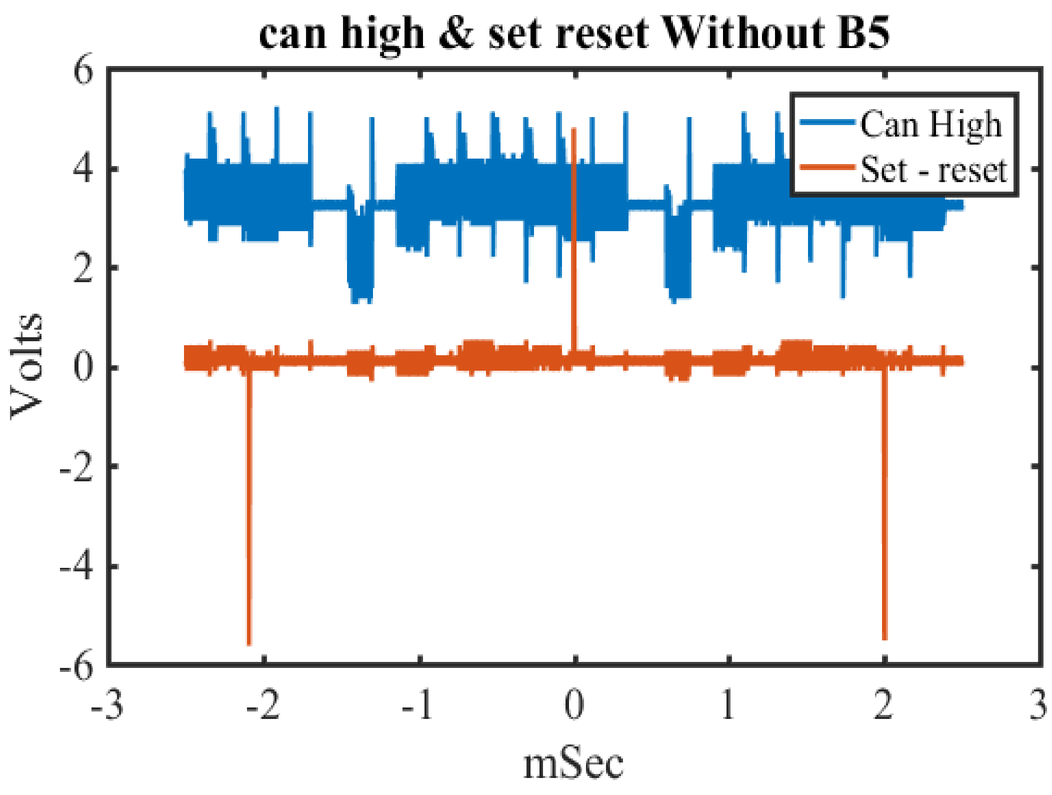


Figure 12.7 : Without B sensors 5

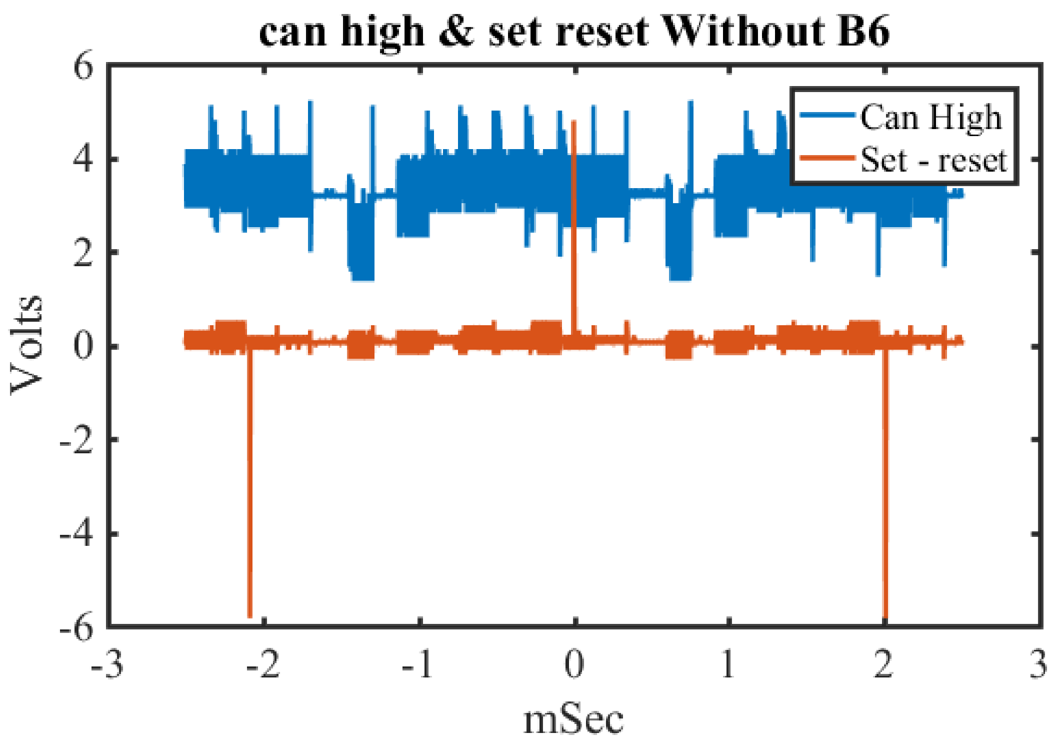


Figure 12.8 : Without B sensors 6

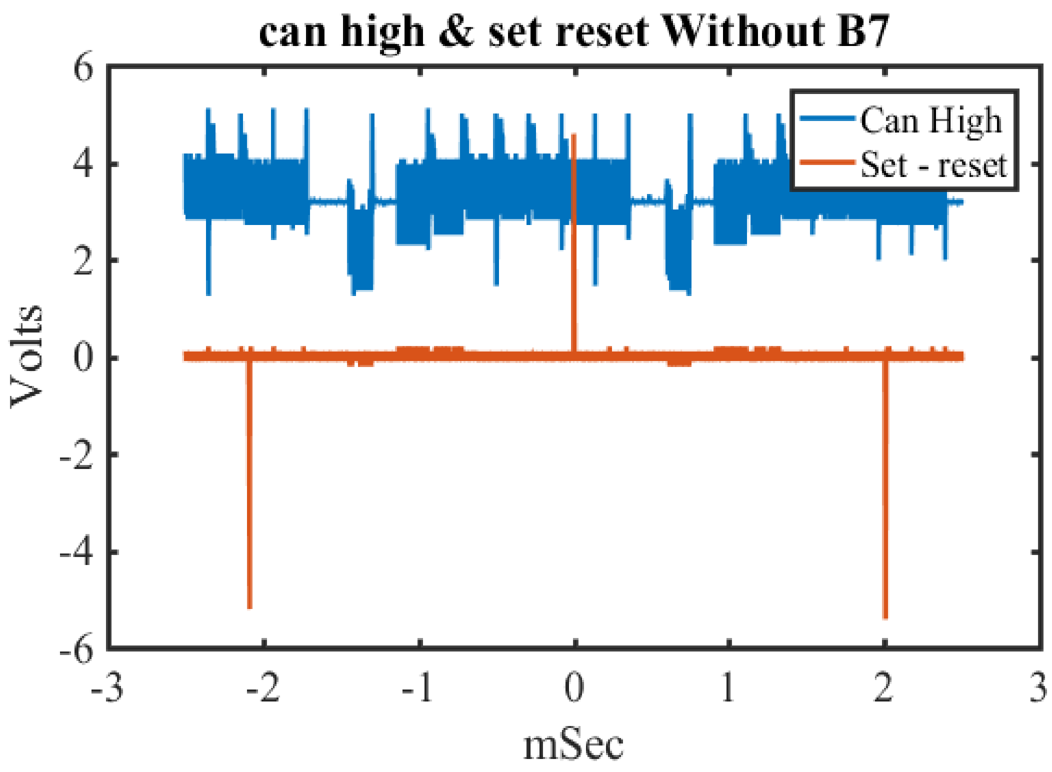


Figure 12.9 : Without B sensors 7

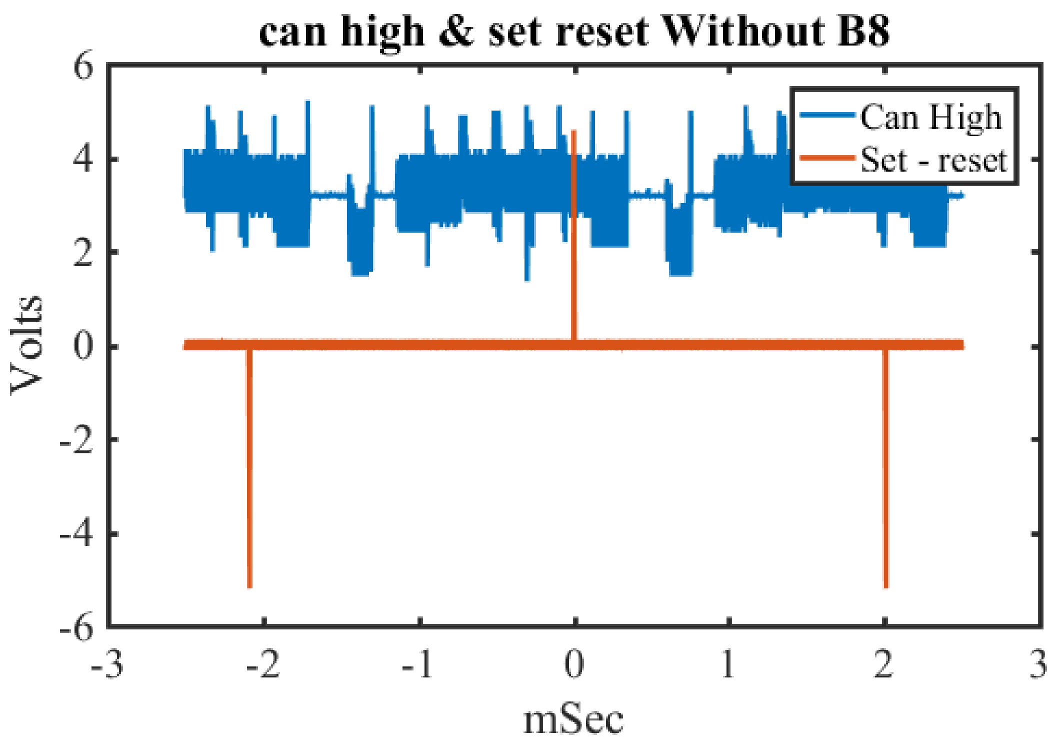


Figure 13 : Without B sensors 8

G.2

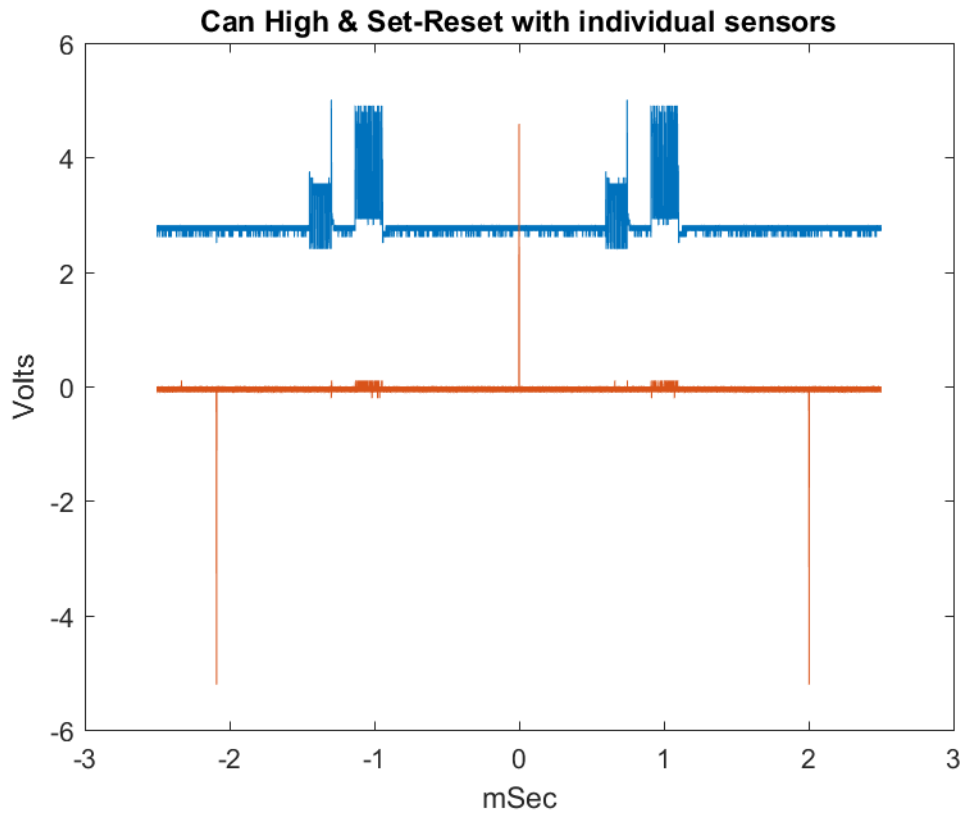


Figure 13.1 : Just B sensor 1

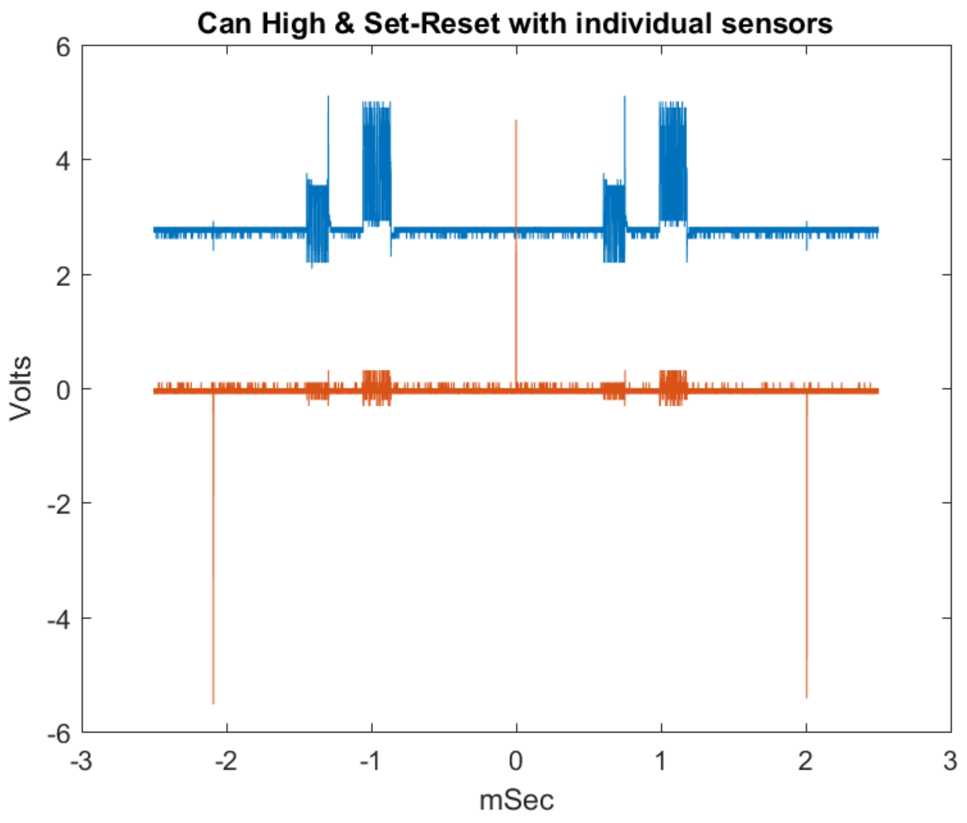


Figure 13.2 : Just B sensor 2

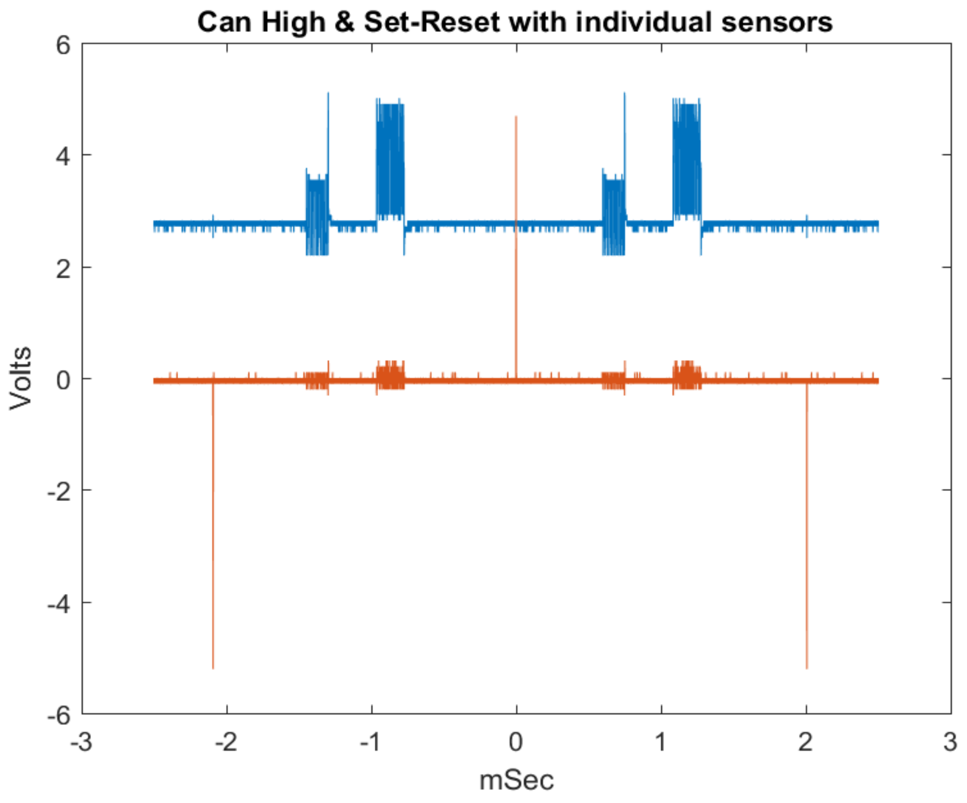


Figure 13.3 : Just B sensor 3

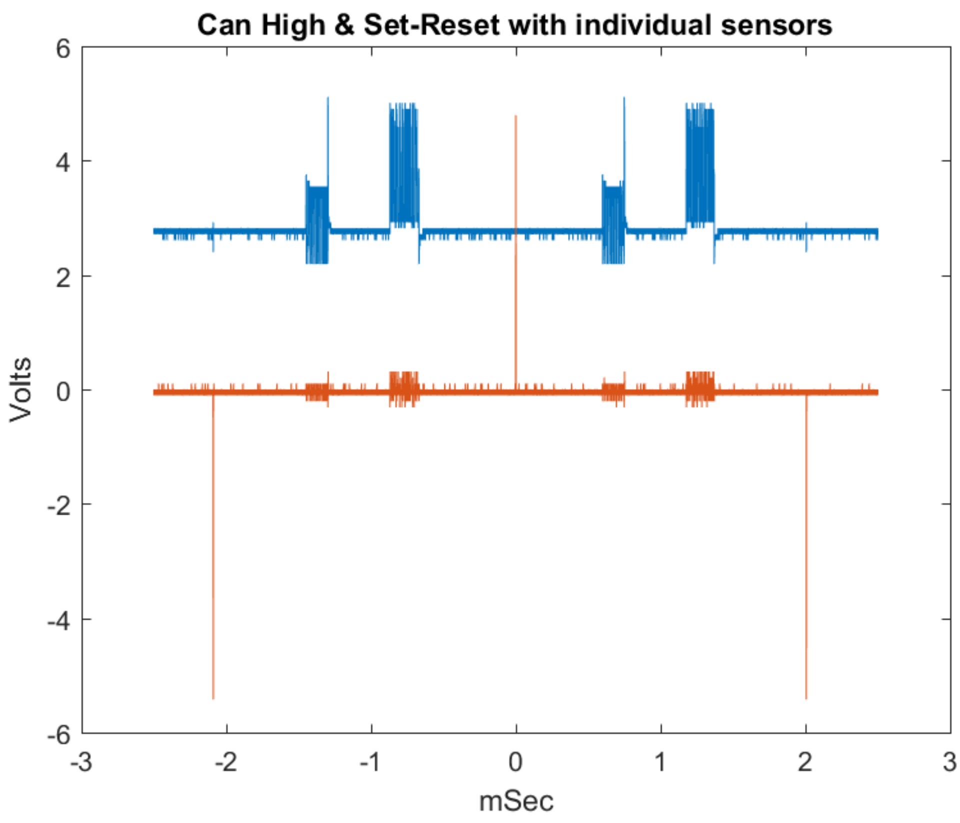


Figure 13.4 : Just B sensor 4

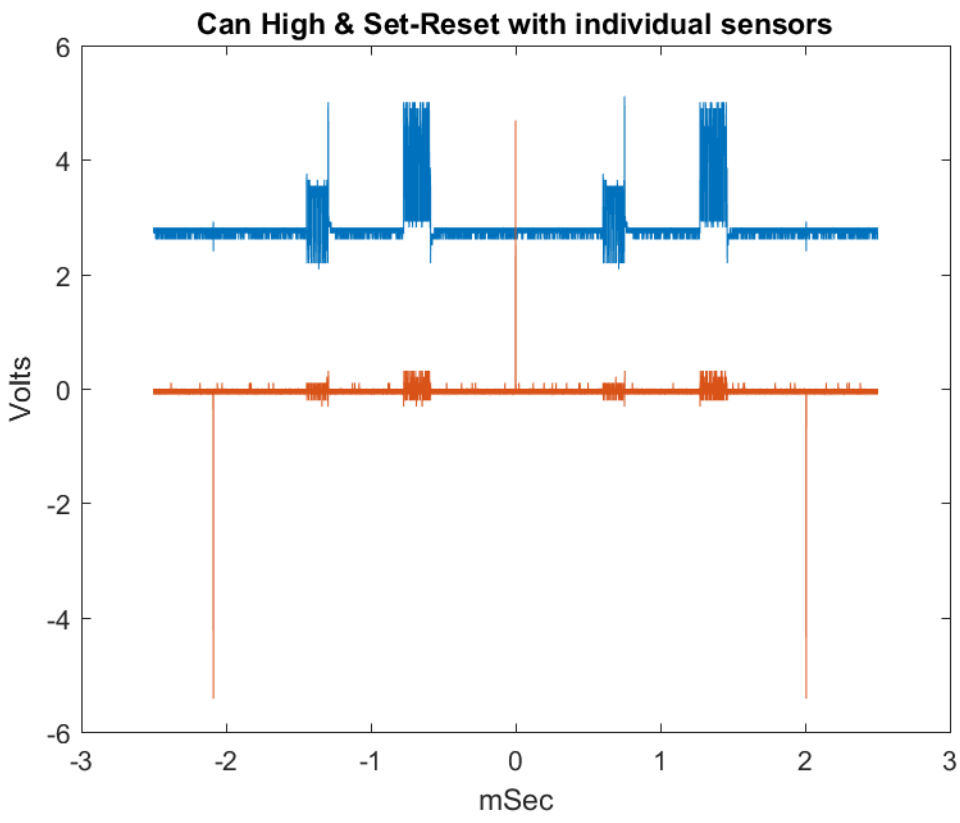


Figure 13.5 : Just B sensor 5

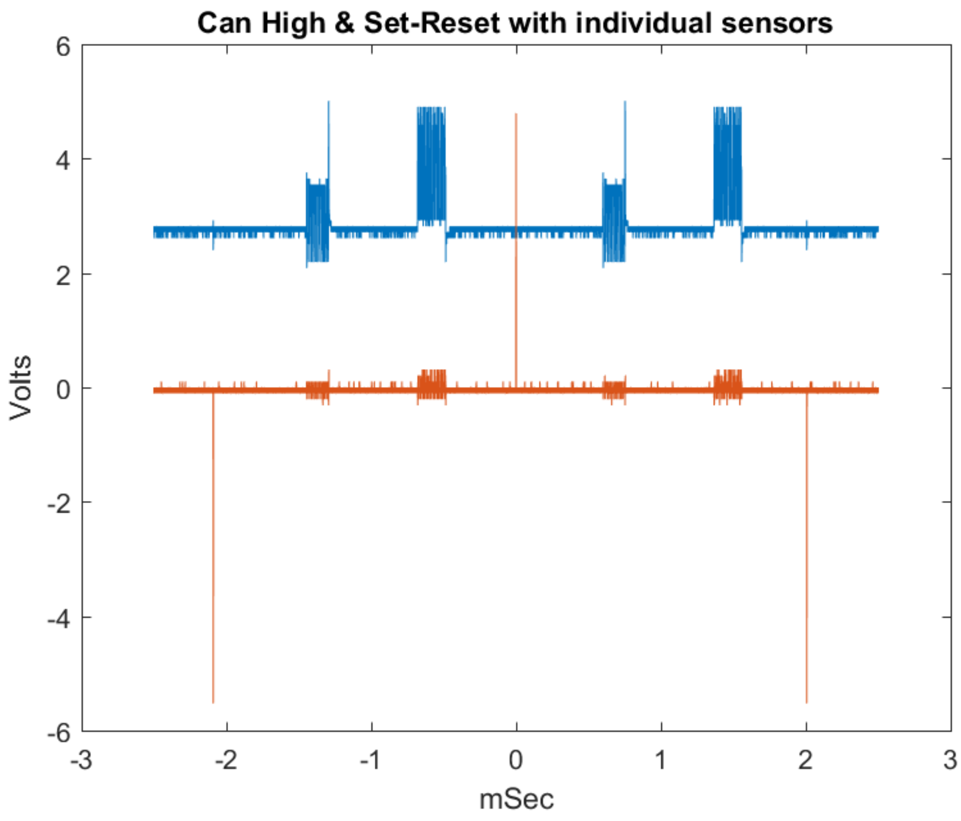


Figure 13.6 : Just B sensor 6

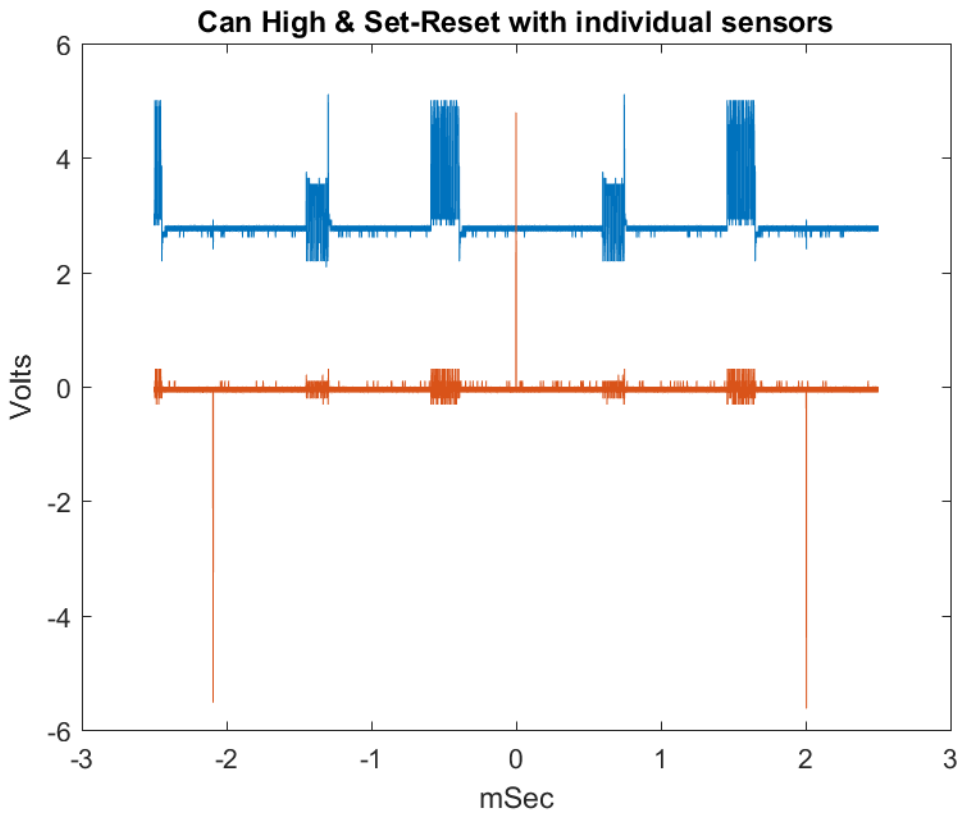


Figure 13.7 : Just B sensor 7

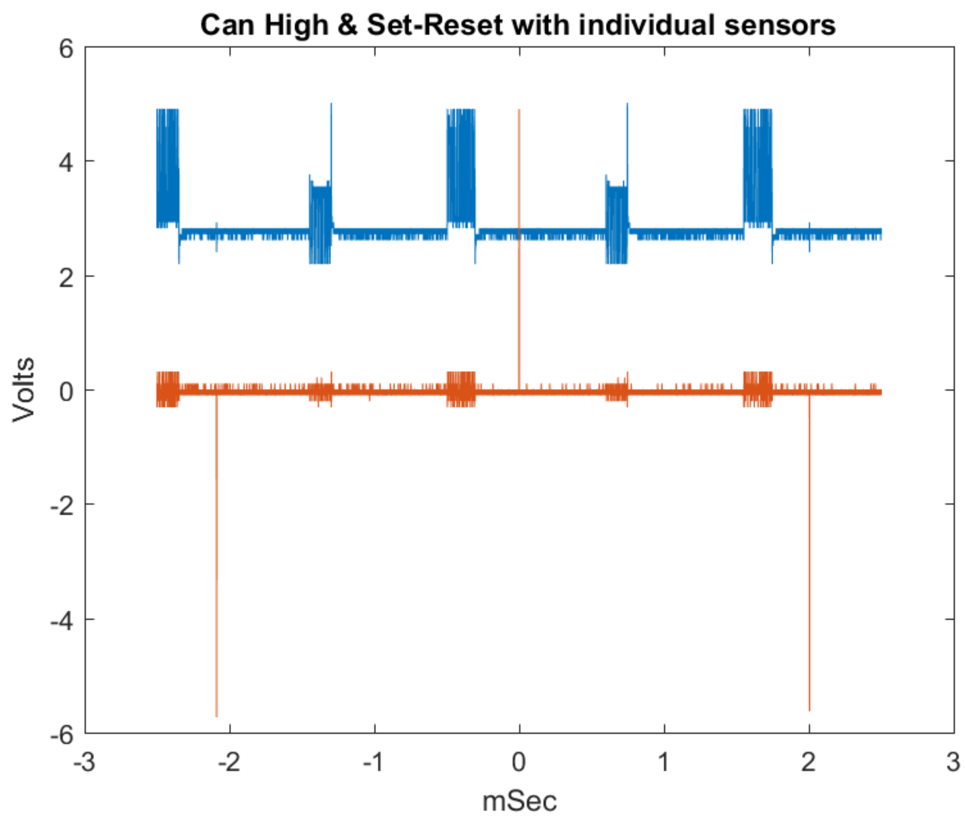


Figure 13.8 : Just B sensor 8

G.3

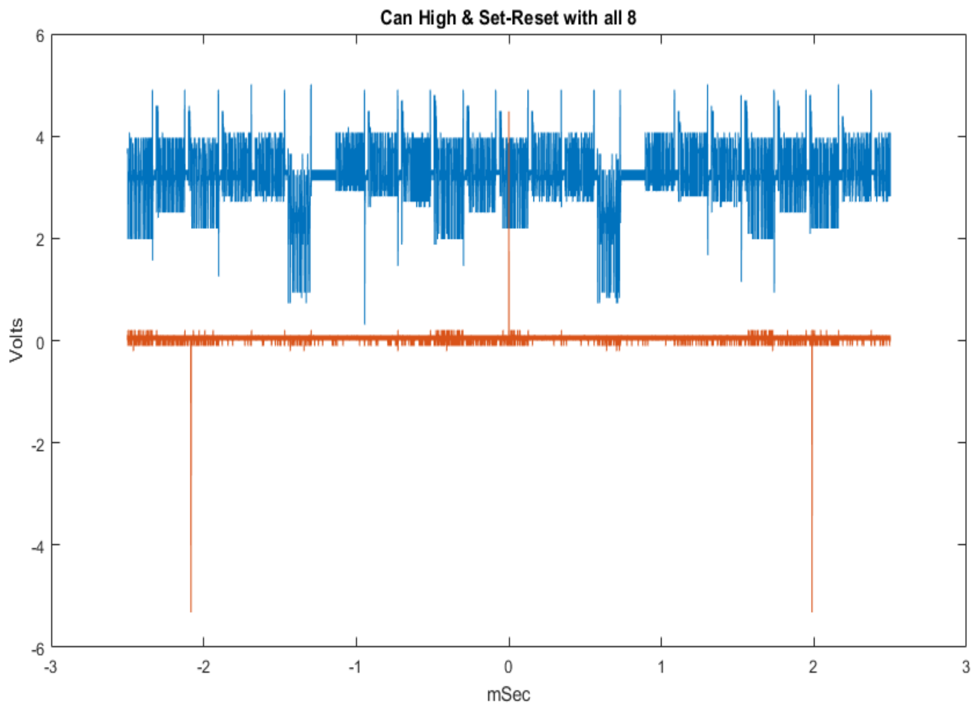


Figure 13.9: all 8 sensors

G.4

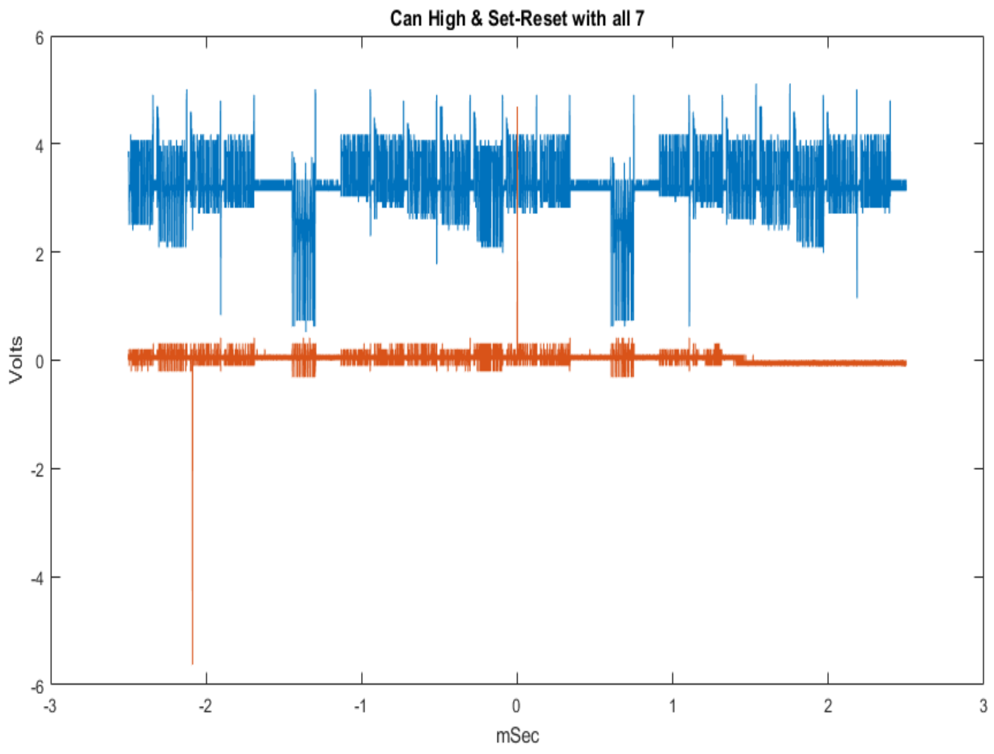


Figure 14 : all 7 sensors

G.5

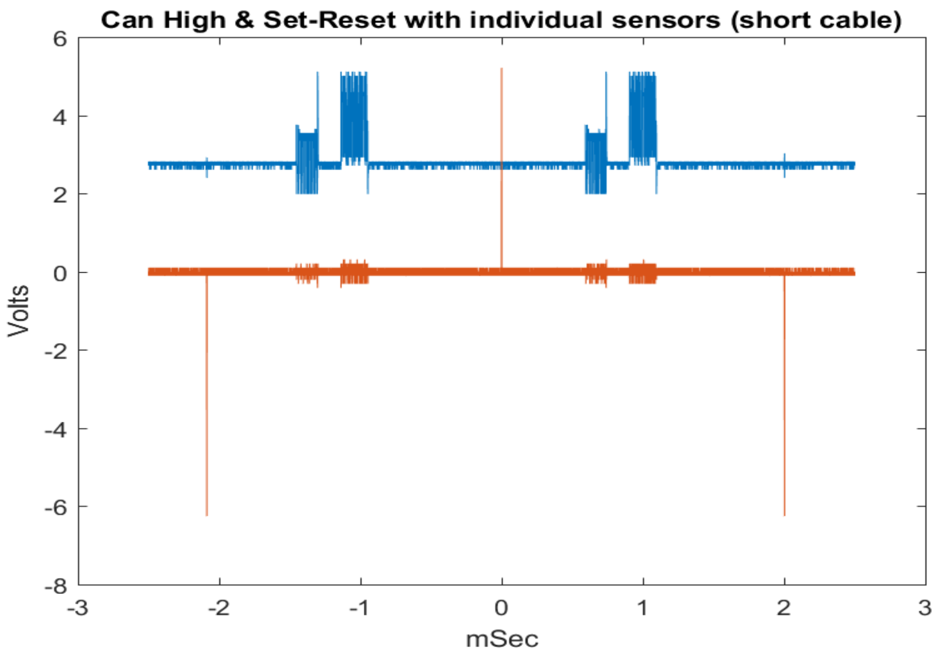


Figure 14.1 : B sensor 1

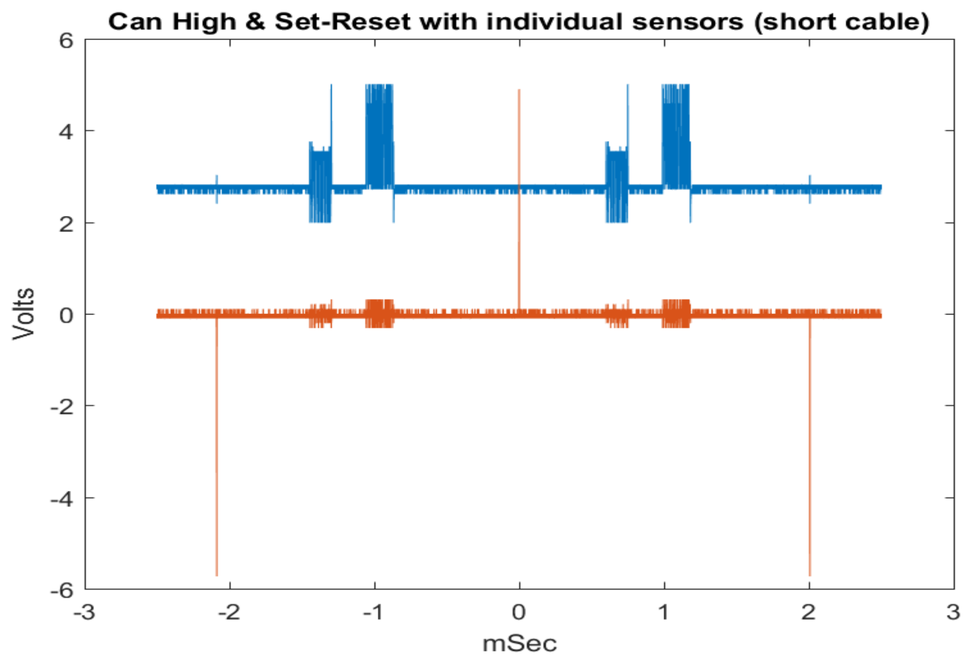


Figure 14.2 : B sensor 2

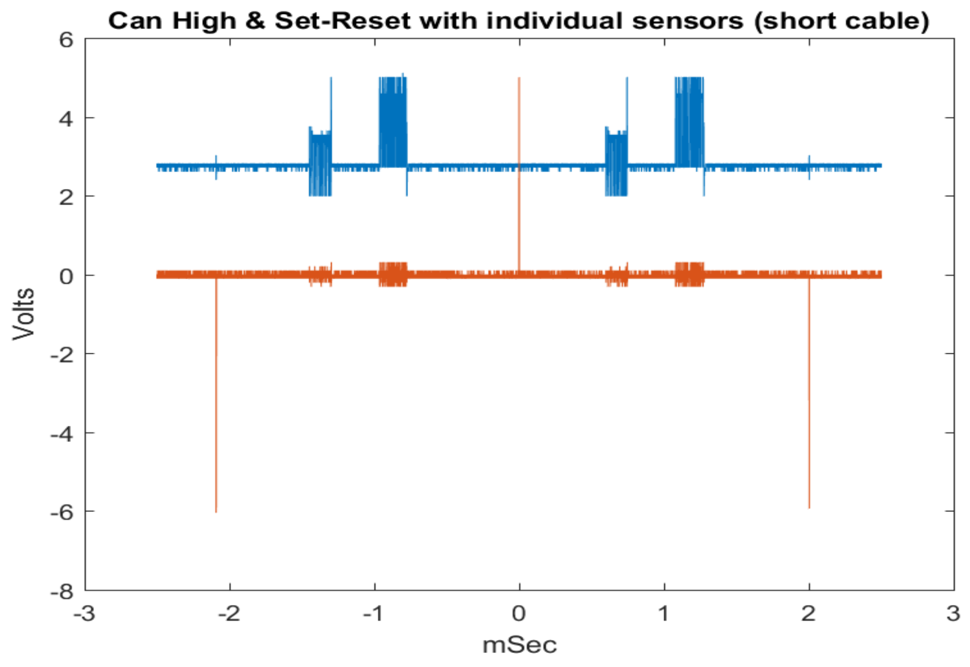


Figure 14.3 : B sensor 3

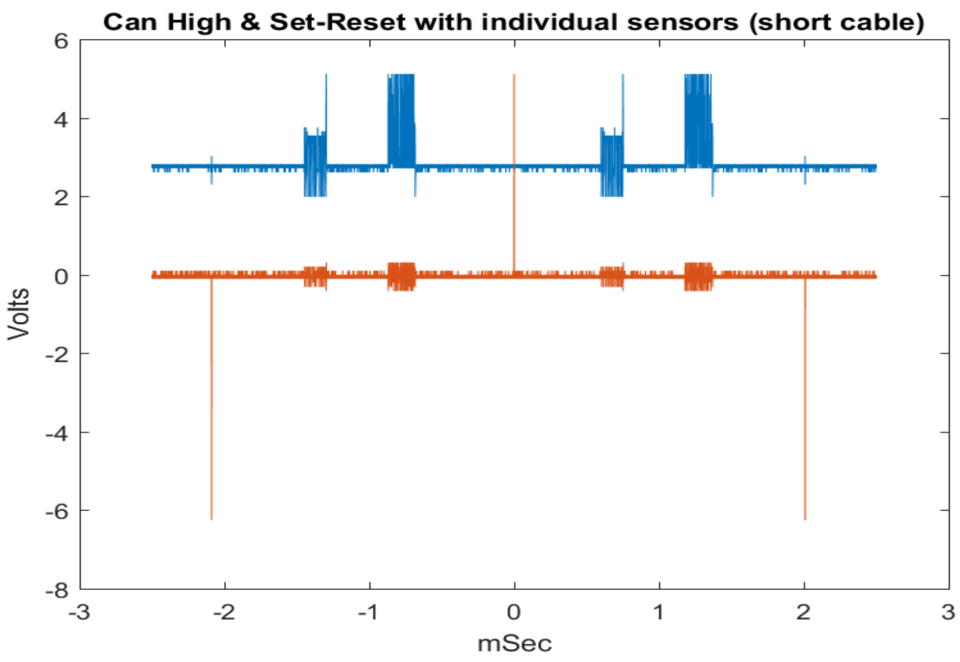


Figure 14.4 : B sensor 4

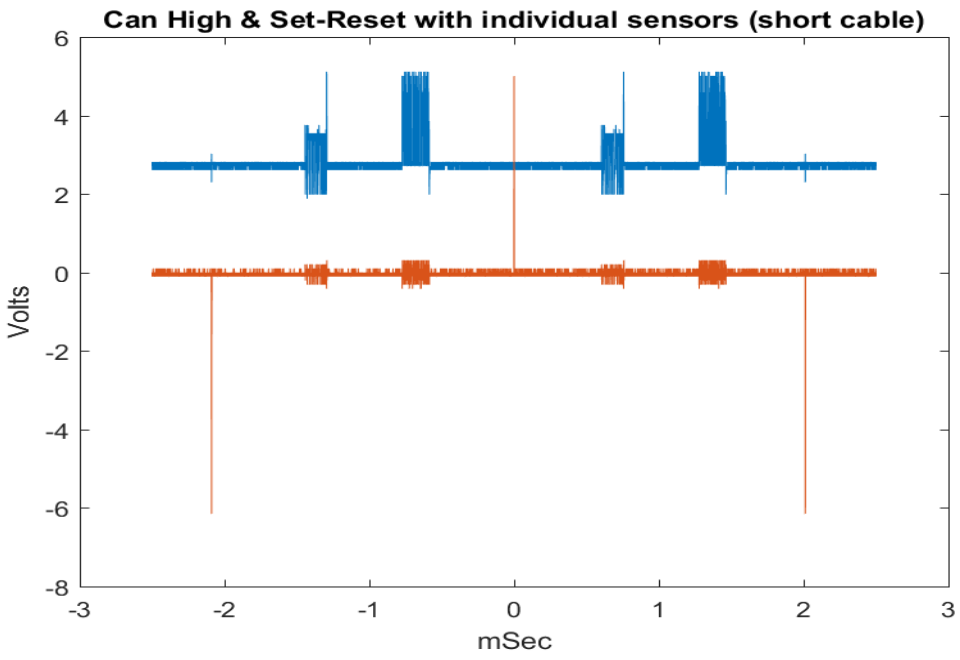


Figure 14.5 : B sensor 5

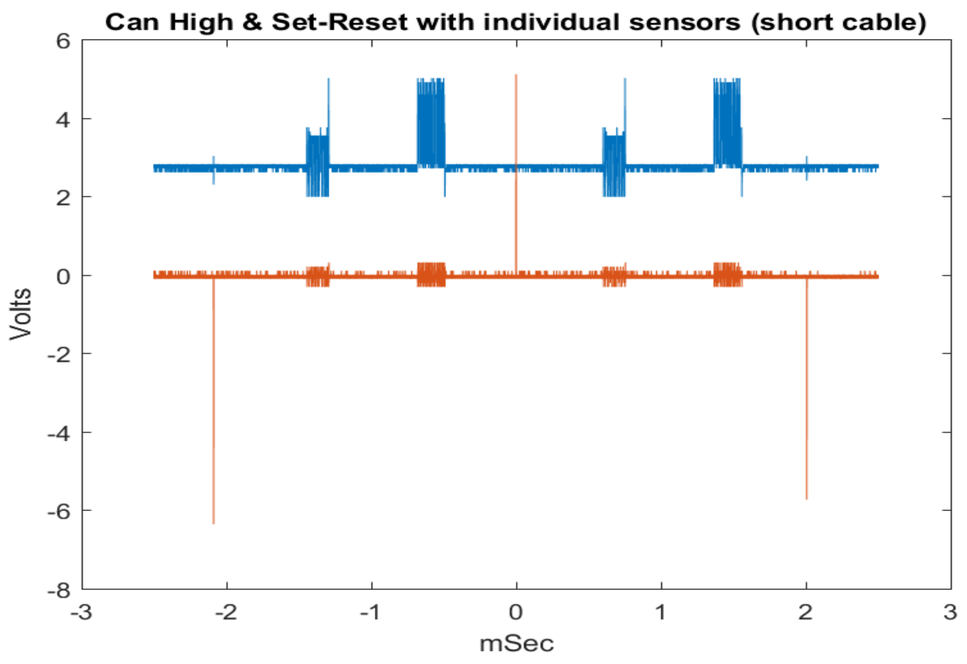


Figure 14.6 : B sensor 6

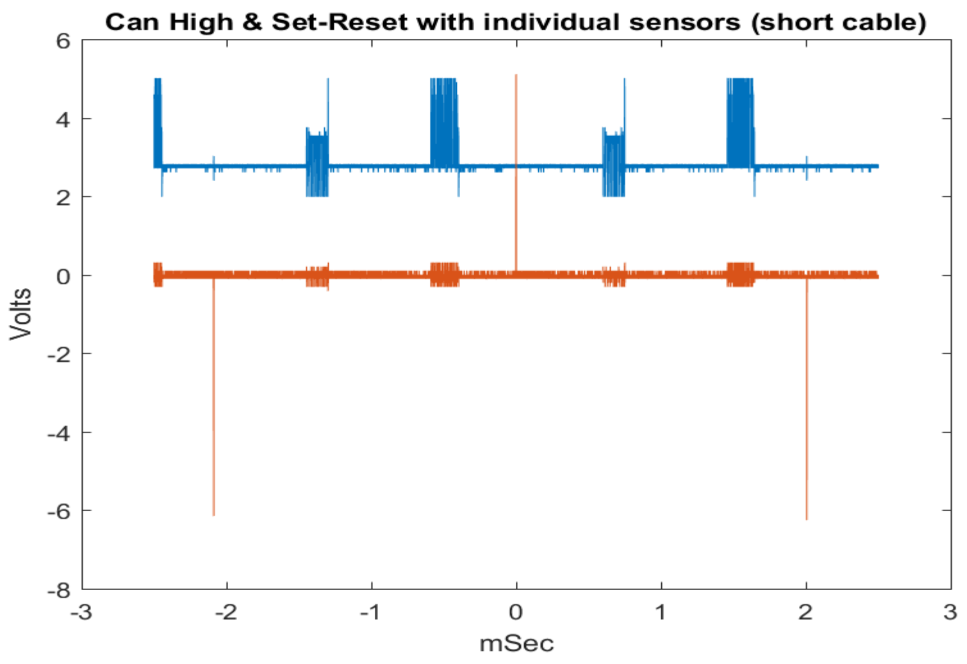


Figure 14.7 : B sensor 7

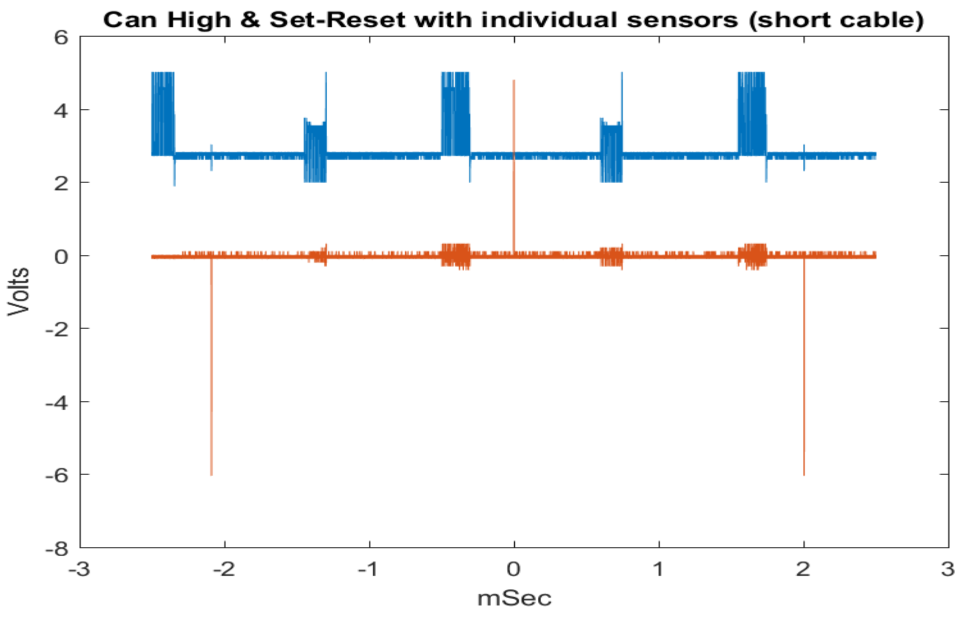


Figure 14.8 : B sensor 8

G.6

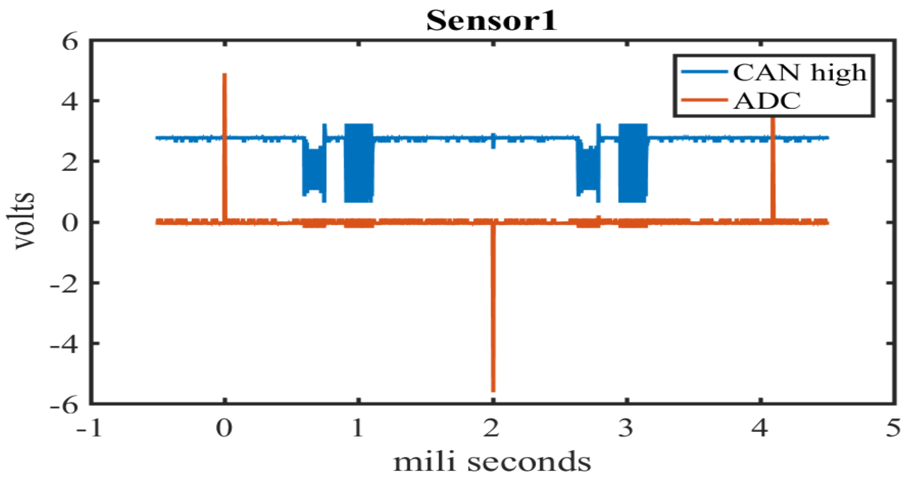


Figure 15.9 : ADC and CAN High, for Sensor 1 with ethernet cable

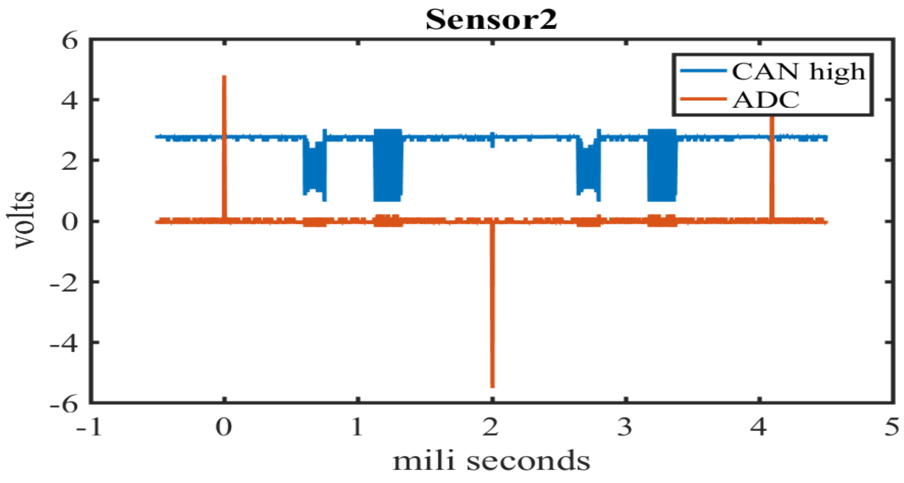


Figure 16 : ADC and CAN High, for Sensor 2 with ethernet cable

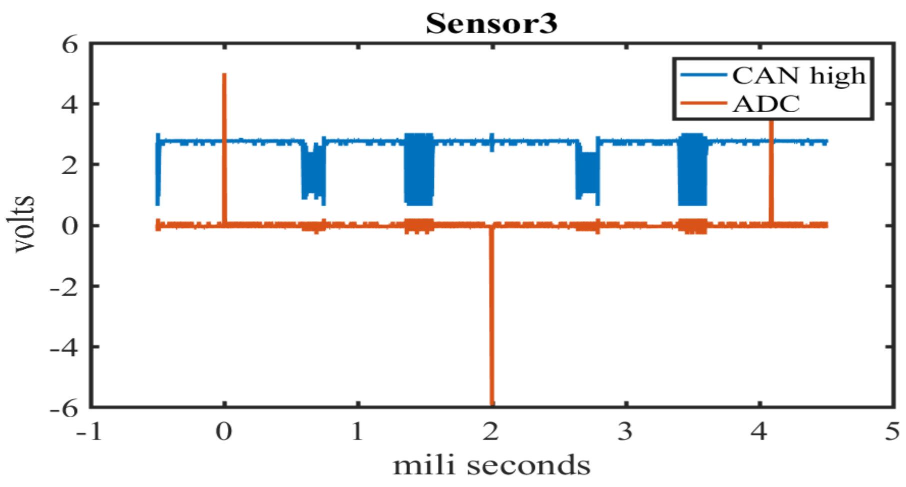


Figure 16.1： ADC and CAN High, for Sensor 3 with ethernet cable

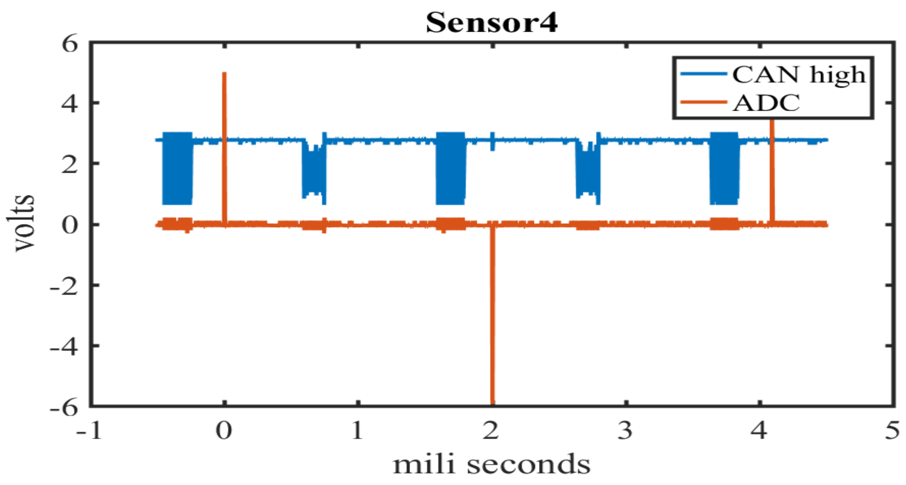


Figure 16.2： ADC and CAN High, for Sensor 4with ethernet cable

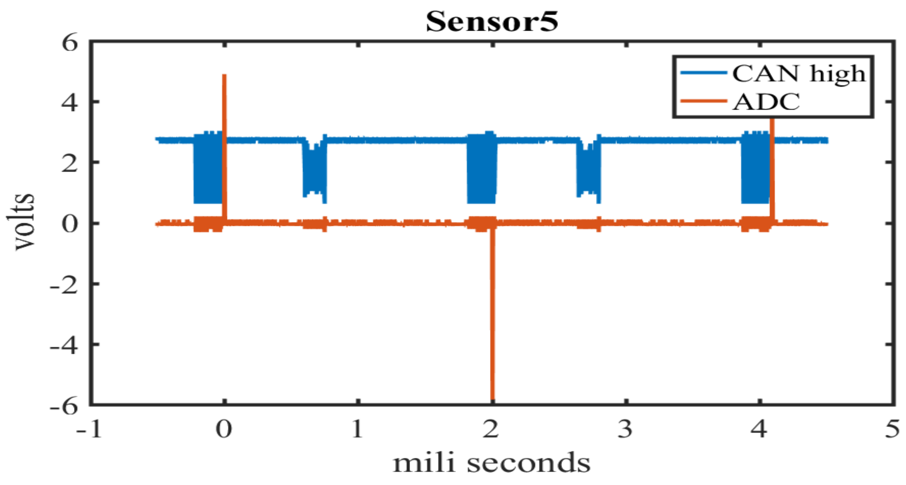


Figure 16.3： ADC and CAN High, for Sensor 5 with ethernet cable

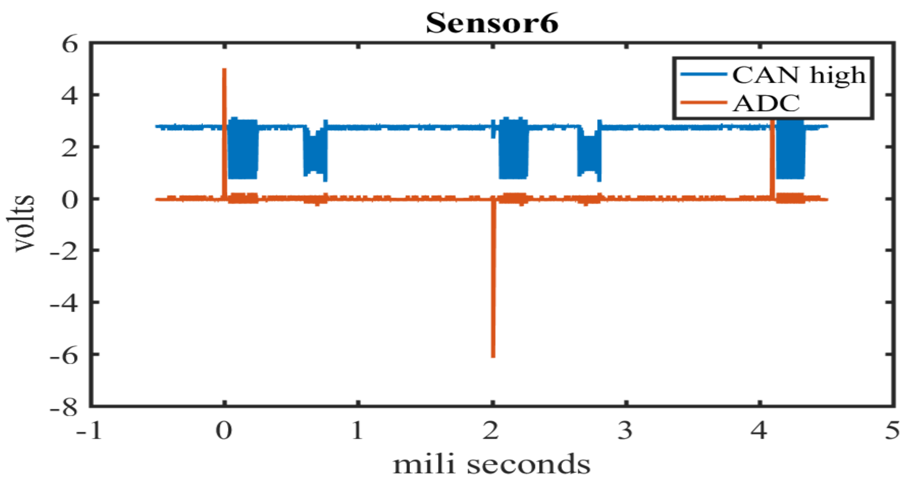


Figure 16.4 : ADC and CAN High, for Sensor 6 with ethernet cable

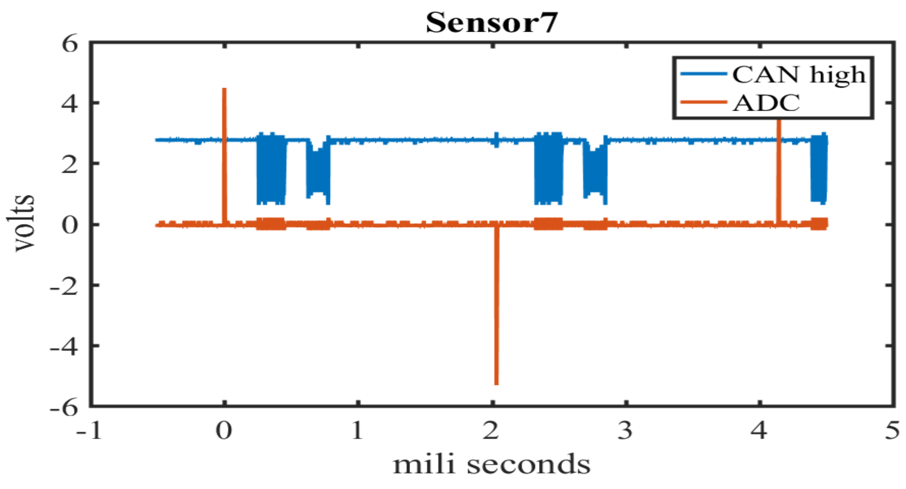


Figure 16.5 : ADC and CAN High, for Sensor 7 with ethernet cable

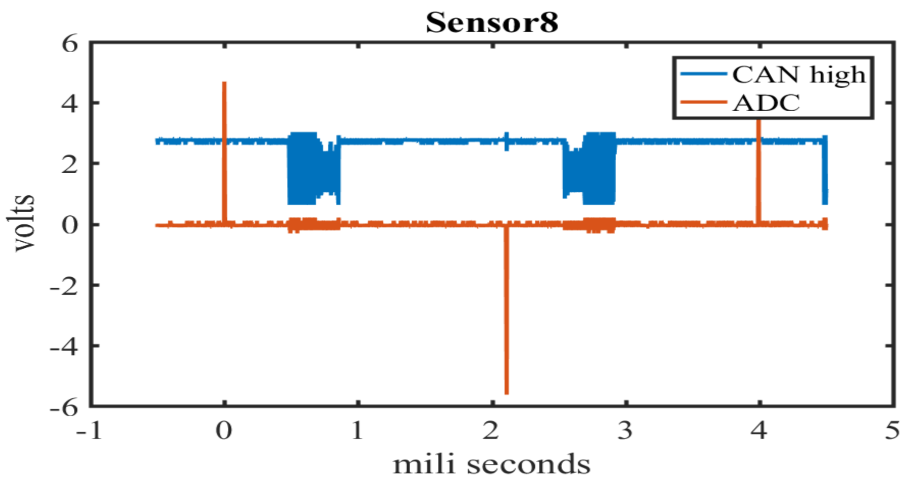


Figure 16.6 : ADC and CAN High, for Sensor 8 with ethernet cable